



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE HERRAMIENTA COLABORATIVA PARA EL LEVANTAMIENTO DE PROCESOS BPMN EN DISPOSITIVOS MÓVILES

**MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN
COMPUTACIÓN**

DIEGO EDUARDO AGUIRRE GONZÁLEZ

**PROFESOR GUÍA:
NELSON BALOIAN TATARYAN**

**MIEBROS DE LA COMISIÓN
ALEXANDRE BERGEL
NELSON BALOIAN TATARYAN
ANDRÉS FARÍAS RIQUELME**

**SANTIAGO DE CHILE
2015**

Resumen Ejecutivo

Hoy en día, el levantamiento de procesos de negocio es un mecanismo ampliamente difundido y utilizado para el entendimiento de reglas, problemáticas y características de empresas y su negocio.

Hoy en día se identifican numerosos problemas asociados al levantamiento de procesos, muchos de los cuales tienen su origen en que el conocimiento sobre los procesos no está formalizado y/o está repartido entre diversos actores. Para enfrentar esto se existen varias técnicas que involucran el trabajo en terreno y la colaboración de varios participantes a la vez.

El trabajo de esta memoria fue la construcción de una herramienta que apoye dichas técnicas, facilitando el trabajo del levantamiento y modelado en terreno y en forma colaborativa.

Fue así como se construyó **MOBIZ**, aplicación que permite el trabajo concurrente de varios actores en el modelamiento de procesos a través de dispositivos móviles.

Al ser construida HTML5, puede ser accedida desde casi cualquier dispositivo moderno común navegador, móvil o no. Entrega una interfaz diferenciada para equipos móviles y de escritorio, y gracias a la creación del framework **COUPLINGSERVER**, se pudo abstraer completamente la aplicación misma de la comunicación con otros usuarios.

Tabla de contenido

| | |
|---|-----------|
| Resumen Ejecutivo | i |
| CAPÍTULO I INTRODUCCIÓN | 1 |
| 1 Motivación | 1 |
| 2 Business Process Management | 3 |
| 3 Levantamiento de procesos de negocios | 5 |
| 3.1 Sobre el levantamiento de procesos..... | 6 |
| 3.2 Mecanismos de levantamiento de procesos | 6 |
| 3.3 NetSketcher | 8 |
| 4 Problemáticas actuales..... | 8 |
| 5 Alcance y objetivos | 9 |
| 5.1 Objetivo general..... | 9 |
| 5.2 Objetivos Especificos..... | 10 |
| 5.3 Resultado esperado..... | 10 |
| 6 Estrategia del proyecto | 11 |
| 6.1 Aplicación móvil | 11 |
| 6.2 Aplicación colaborativa | 12 |
| 6.3 Interfaz simple y flexible, pero completa | 12 |
| CAPÍTULO II MOBIZ: APLICACIÓN BPMN..... | 15 |
| 7 Introducción, alcance y objetivo..... | 15 |
| 8 El modelador BPMN: Mobiz..... | 16 |
| 9 Principios de Diseño | 16 |
| 9.1 HTML5 | 16 |
| 9.2 Doble Interfaz..... | 17 |
| 10 Tecnologías a utilizar | 18 |
| 11 Diseño de la solución..... | 18 |
| 11.1 Diseño general..... | 18 |
| 11.2 Diseño de interfaces | 23 |
| 11.3 Detalles de la implementación | 33 |
| CAPÍTULO III FRAMEWORK DE COMUNICACIONES | 40 |
| 12 Introducción, alcance y objetivos | 40 |
| 13 Objetos Acoplados..... | 41 |
| 14 Implementaciones anteriores..... | 42 |

| | | |
|--|--|-----------|
| 14.1 | MatchMaker RMI..... | 42 |
| 14.2 | MatchMaker SOAP | 43 |
| 14.3 | Objetos Acoplados en HTML5 | 43 |
| 15 | Implementación propuesta | 44 |
| 16 | Diseño de la solución..... | 46 |
| 16.1 | API propuesta | 46 |
| 16.2 | Tecnología | 47 |
| 16.3 | Conceptos generales | 47 |
| 16.4 | El servidor | 48 |
| 16.5 | El cliente | 56 |
| 17 | Transformación de Mobiz en una aplicación concurrente..... | 60 |
| 17.1 | Modificaciones necesarias | 62 |
| 17.2 | Acceso a un modelo en particular..... | 63 |
| CAPÍTULO IV CONCLUSIÓN Y TRABAJOS FUTUROS | | 64 |
| 18 | Análisis de Resultados | 64 |
| 18.1 | Funcionalidades e interfaz | 64 |
| 18.2 | Framework de comunicación | 65 |
| 19 | Conclusión..... | 66 |
| 20 | Trabajos Futuros | 68 |
| CAPÍTULO V REFERENCIAS | | 70 |
| 21 | Bibliografía..... | 70 |
| CAPÍTULO VI APÉNDICES | | 72 |
| 1 | Anexo A. Características y Comparación entre Canvas y SVG..... | 72 |
| 1.1 | Canvas..... | 72 |
| 2 | Anexo B. Compatibilidad de Browser con HTML5..... | 73 |
| 3 | Anexo C. Detalles de implementación CouplingServer | 76 |

Capítulo I

Introducción

1 Motivación

Como nunca antes, las tecnologías de información (TI) y las estrategias de las organizaciones se encuentran en condiciones de colaborar de manera eficaz y eficiente. Esto se debe principalmente a que tanto el área de TI como el resto de la organización comparten una visión y paradigma similar y complementario: los *procesos de negocio*. Con esta visión en común, el área de TI busca formar una biblioteca de servicios que de un apoyo tecnológico a las actividades de los procesos de negocio. De una manera equivalente, la organización, o notablemente el *área de procesos*, definirá sus procesos de negocio teniendo en cuenta las actividades reales que ejecuta la organización, organizándolas como piezas reutilizables que serán el pilar que permitan entender la dinámica operacional de la organización.

En este contexto, el área de TI trabaja de la mano con otras áreas de la organización para definir los *procesos de negocio* de la organización utilizando un lenguaje en común: BPMN, acrónimo inglés de *Business Process Modeling Notation*. En la actualidad, las organizaciones de nuestro país que se encuentran adoptando este enfoque realizan normalmente un proceso de descubrimiento de sus procesos, que consiste principalmente en describir los principales procesos existentes de la organización con esta notación. En algunos casos existen individuos en la organización que poseen el conocimiento necesario del negocio para poder realizar el levantamiento, mientras que

en otros casos es necesario reunir a diversos actores para poder tener una visión completa de un proceso. Este último caso es generalmente el escenario más frecuente para las organizaciones que cuentan con procesos de negocio complejos donde intervienen diversos actores, generalmente pertenecientes a distintas áreas.

El levantamiento de los procesos de negocio será aun más complejo cuando quienes posean los conocimientos relativos a dichos procesos se encuentren diseminados a lo largo de la organización. Si la organización además tiene una presencia en distintas ubicaciones geográficas el problema se verá acentuado. No sólo la cantidad de actores y su ubicación geográfica serán factores que determinen la dificultad para el levantamiento de los procesos de negocio, sino también otros factores que dependen de los participantes del proceso, tales como:

- **Visión del proceso.** Los individuos que participan en un proceso suelen tener distintas visiones del mismo proceso. Esto por supuesto conlleva a un verdadero descubrimiento del proceso, y en muchos casos extiende la duración de esta actividad debido a las múltiples iteraciones que se producen debido al encuentro de diversas visiones.
- **Distintos intereses.** Generalmente los actores de un proceso de negocio tienen intereses definidos y localizados particularmente en las responsabilidades que les atañe con respecto a las actividades que realizan, las que pueden no corresponder a los intereses de la organización.

En dichos escenarios es deseable que todos los actores puedan participar concurrentemente en el levantamiento del proceso a fin de poder minimizar las iteraciones sobre su definición. Con esta actividad todos los actores se enriquecen de la perspectiva de los otros participantes. Dado que los actores pueden encontrarse en lugares geográficos distintos, la solución idónea consiste naturalmente en una herramienta de trabajo colaborativo con las siguientes características:

- **Soporte de notación BPMN 2.0.** Existen diversas maneras de documentar un proceso, así como diversos lenguajes en los cuales se puede realizar la documentación. Hoy en día el lenguaje más utilizado para la documentación de procesos de negocio es **BPMN 2.0**. Este lenguaje gráfico permite documentar los procesos utilizando un lenguaje no formal, pero claro y con diversas notaciones para expresar todo tipo de dinámicas en los procesos.
- **Edición colaborativa de un diagrama.** Una de las problemáticas más importantes a enfrentar en un escenario como el descrito previamente es precisamente la habilidad de poder hacer colaborar a diversos participantes, incluso a veces ubicados en lugares distintas, simultáneamente. Más aún, y de acuerdo con lo indicado en el punto anterior, esta habilidad debe ser propia de la herramienta que permita describir el proceso utilizando la notación **BPMN 2.0**.

- **Soporte en Web.** La instalación de plataformas para la utilización de un sistema o una aplicación en particular es claramente una barrera para su uso. Los navegadores web, por otro lado, son aplicaciones que funcionan también como ambientes de ejecución y además se encuentra instalado en la gran mayoría¹ de los computadores en el mundo.
- **Soporte en web para dispositivos móviles.** No todos los procesos de negocio ocurren en las dependencias de la organización: muchos de ellos, o al menos algunas de sus actividades pueden estar ocurriendo en terreno, y por ende no será siempre factible acceder a un navegador web completo. Es por esta razón que se vuelve fundamental poder contar con una herramienta que no sólo sea accesible a través de un navegador, sino también a través de navegadores más limitados como los que se encuentran instalados en dispositivos de telefonía móviles (teléfonos que tienen un sistema operativo instalado tal como **IOS**, **ANDROID**, **BLACKBERRY**, etc.). Con esto, quien realice el levantamiento de procesos puede desplazarse por y entre las dependencias, entrevistando a sus actores y observando el proceso.

2 Business Process Management

BUSINESS PROCESS MANAGEMENT (BPM) es un enfoque para describir y documentar *procesos de negocios* como activos de una organización, administrando sus ciclos de vida y observándolos para mejorarlos y optimizarlos. Usando algunos de los beneficios claves de BPM, Gartner (Gartner, 2005) lo define como algo que *da control de los procesos de negocio para mejorar la agilidad y el rendimiento operacional* (Buelow, Das, Deb, Palvankar, & Srinivasan, 2010).

Según (Buelow, Das, Deb, Palvankar, & Srinivasan, 2010) *el principal aporte de adoptar BPM está en el obtener excelencia organizacional de modo de obtener ventajas sobre la competencia. Esto se logra al alinear las necesidades del negocio con sus procesos, actores y las tecnologías que le dan soporte.* Según Buelow et al. (Buelow, Das, Deb, Palvankar, & Srinivasan, 2010) los principales beneficios de adoptar BPM se pueden resumir según la siguiente Tabla 1:

¹ Es evidente que no es posible afirmar que todos los computadores del mundo poseen un navegador web instalado, pero en la práctica, esta afirmación es cierta para todos los computadores que son utilizados por usuarios finales.

Tabla 1: Beneficios principales de adoptar BPM

| | Efficiency | Visibility | Agility |
|----------------|---|--|--|
| | Better, faster and more cost effective than your current alternative | Know the current status and outcome of your processes & business | Adapt quickly to changing business conditions |
| Metrics | <ul style="list-style-type: none"> • Utilization, capacity • Throughput, speed • Quality, yield, exceptions | <ul style="list-style-type: none"> • Financial • Organizational • SLA failure rate • Rate of non-compliance | <ul style="list-style-type: none"> • Speed to create & change processes • Time to market |
| Results | <ul style="list-style-type: none"> • Reduced Cost • Improved productivity/ROI • Effective resource utilization • Better quality / service | <ul style="list-style-type: none"> • Managed, lower risk • Compliance • Financial accountability • Lower capital reserves • Better visibility | <ul style="list-style-type: none"> • New revenue growth • Market share growth • Increased competitiveness • Thought leadership |

Es por esto que es cada vez es más común ver empresas que se encuentran realizando un esfuerzo por adoptar **BPM** en búsqueda de obtener los beneficios que esto ofrece. Sin embargo, esto no está carente de riesgos, pues la adopción de **BPM** es un proceso complejo que muchas veces, por no ser abordado adecuadamente, puede fracasar con altos costos para las empresas (Buelow, Das, Deb, Palvankar, & Srinivasan, 2010). Adoptar **BPM** no es sólo identificar, modelar y automatizar los procesos, sino que requiere cambios a nivel organizacional, metodológico y tecnológico como se ilustra en la Figura 1.



Figura 1. Áreas relevantes para la adopción BPM.

De esta forma, una cuidadosa planificación de los procesos, y del proceso de adopción en sí es clave para el éxito. Para poder hacer esto, lo primero que un analista debe hacer es intentar entender y modelar los procesos de negocio de la organización. Esta etapa se conoce como el *levantamiento de procesos de negocios*.

3 Levantamiento de procesos de negocios

Como se ha mencionado anteriormente, la adopción de un paradigma BPM por una organización comienza y luego continúa siempre con el levantamiento de procesos de negocio. A continuación discutiremos acerca del proceso de levantamiento en sí, mientras que a continuación veremos algunas técnicas conocidas y populares para realizar esta tarea. Finalmente, en esta sección revisaremos una herramienta desarrollada en el **DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN (DCC)** de la **UNIVERSIDAD DE CHILE** denominada **NETSKETCHER**. Esta herramienta servirá como un punto de partida para una implementación que permita satisfacer las condiciones impuestas por un proceso que se realiza en organizaciones donde el conocimiento sobre el proceso se encuentra distribuido en diversos participantes, ubicados en distintos lugares geográficos.

3.1 Sobre el levantamiento de procesos

Parte fundamental de **BPM** es el *levantamiento de procesos de negocios*, que consiste en primer lugar en entender las actividades involucradas y luego identificar sus entradas, salidas, flujo de las actividades, puntos de decisión, paralelismos y convergencias, dando así forma a un flujo de control (Baloian, y otros, 2011).

Para poder llevar esto a algún modelo, es necesario usar algún tipo de notación. Para esto existen dos notaciones que son utilizadas ampliamente en la actualidad: **BPEL** (acrónimo del inglés *Business Process Execution Language*) y **BPMN** (Buelow, Das, Deb, Palvankar, & Srinivasan, 2010).

- **BPEL** es un lenguaje basado en XML. Como su nombre lo sugiere, es principalmente útil para ser consumidos por motores de ejecución de procesos, como el incluido en Oracle BPM Suite entre otros.
- **BPMN** es una notación grafica para modelar procesos de negocios. Esta notación es la que más se utiliza actualmente por los analistas de procesos.

El *levantamiento de procesos de negocio* es actualmente considerado una herramienta muy útil no sólo para proyectos de *adopción BPM*, sino para cualquier proyecto que involucre mejorar los procesos de negocios, redefinición de estrategias de negocios y para el levantamiento de requerimientos de sistemas (Baloian, y otros, 2011). Existen plataformas que usan los modelos de los procesos como la base para el desarrollo de sistemas de información que apoyen los procesos de negocio, como **ORACLE BPM** o **IBM BPM**.

Una fuente importante de problemas en el levantamiento de procesos radica en que el conocimiento del proceso esta diseminado entre los agentes involucrados en el procesos, ya sea *actores directos* o *actores indirectos*, dificultando la obtención y consolidación de la información necesaria para levantar el proceso (Gonçalves, Santoro, & Baião, 2010). Si bien existen técnicas que se han mostrado efectivas en la obtención y consolidación de información, aún se observan problemas a los que se ve enfrentado quien realiza esta tarea (Wang, Zhao, & Zhang, 2009).

Según De Laurentiis, en su metodología **BPM:RAD**, el levantamiento de los procesos de negocio, debe hacerse tan rápido y eficiente como sea posible, para poder mantener las expectativas del cliente (Laurentiis, 2011), lo que hace aún más compleja la tarea.

3.2 Mecanismos de levantamiento de procesos

Existen diversas barreras y dificultades que han sido identificadas en el contexto del *levantamiento de procesos de negocio* (Baloian, y otros, 2011). La mayoría de las problemáticas identificadas se dan a lugar debido a la complejidad de las organizaciones y de las relaciones entre sus partes, así como a lo incompleta que es la

información que manejan los actores del proceso, los cambios que ocurren en las organizaciones y la ambigüedad entre la información entregada por diversos actores. Los factores de problemas más comunes en el levantamiento de procesos son:

- **Entendimiento parcial.** Los responsables de los procesos pueden no entender por completo el proceso, por lo que es necesario que varios participen.
- **Comprensión.** Los analistas no comprenden bien el negocio por lo que no pueden hacer preguntas adecuadas.
- **Acceso a información de procesos.** No tienen acceso a otra información relevante como modelos ya construidos o anotaciones de otras entrevistas. El analista puede no recolectar la información de todas las fuentes relevantes.
- **Vocabulario de negocio.** Diferencias de vocabulario entre los analistas y los responsables de los procesos.
- **Técnicas y metodologías.** El analista escoge técnicas de levantamiento de procesos de acuerdo a su experiencia (Bhaumik & Rajagopalan, 2009).

Por otro lado, Verner plantea que el conocimiento de los procesos es *tácito* (Verner, 2004), pues existe en la mente de quienes participan, y *local*, pues cada participante tiene su visión propia del proceso. Además, se plantea que el conocimiento del proceso es esencialmente visual, por lo que cualquier técnica utilizada debe estar basada en elementos que consideren esta característica.

Por estas mismas razones Bhaumik plantea que es recomendable que los analistas de procesos utilicen no una, sino varias técnicas de levantamiento, dependiendo del contexto del proyecto (Bhaumik & Rajagopalan, 2009).

Es entonces relevante el tener en cuenta varias técnicas, tanto *tradicionales* como *colaborativas*. Las primeras son usualmente ejecutadas por un analista o consultor quien realiza entrevistas y workshops a los encargados de los procesos para obtener la información. Las *colaborativas* involucran la participación de varias personas, entre *actores* de los procesos, *encargados* y *analistas*, que interactúan entre ellos para ir recabando el conocimiento sobre las actividades del proceso (Baloian, y otros, 2011).

Si bien las entrevistas y talleres de trabajo con los encargados de los procesos (conocidos mejor por su palabra en inglés *workshops*) siguen siendo las técnicas más utilizadas (Bhaumik & Rajagopalan, 2009), existen propuestas de técnicas que buscan complementarlas con participación de los actores del proceso y expertos del negocio en la construcción de los modelos mismos (Grosskopf, Edelman, & Weske, 2009), (Hoppenbrouwers & Schotten, 2009).

3.3 *NetSketcher*

NETSKETCHER (Baloian, y otros, 2011) es una aplicación que busca facilitar la tarea del levantamiento de procesos de negocios y la construcción de los modelos asociados usando **BPMN**. Esta herramienta se basa en tres principios de diseño que discutimos a continuación:

3.3.1 *Movilidad*

Los autores de **NETSKETCHER** plantean que la aplicación debe ser de carácter móvil, pues hacerlo presenta una serie de ventajas entre las que destacan:

- Permite hacer las entrevistas en el lugar de trabajo de los entrevistados, permitiendo que estos estén más cómodos y que tengan acceso a más información que originalmente no pensaba necesitar o de difícil transporte.
- Permite mayor versatilidad en la conversación, pudiendo incluso cambiar de ubicación si se necesita para, por ejemplo, ir a hablar con otro actor o ver alguna actividad en funcionamiento.

3.3.2 *Colaborativo*

La aplicación debe permitir compartir los modelos **BPMN** entre los analistas en forma síncrona y asíncrona. Esto permite que un analista tenga a su disposición en la información recabada por otros analistas en otras entrevistas y de esta forma poder, por ejemplo, hacer preguntas relevantes que no habría ideado de otra forma.

3.3.3 *Interfaz tipo lápiz y papel*

Se desea que la aplicación permita el ingreso de la información sobre el proceso recabada de una entrevista directamente sobre un modelo usando **BPMN**. Esto implica que debe tener una interfaz suficientemente simple para que hacerlo no entorpezca la reunión. Los autores concluyen que una buena manera de hacer esto es mediante una interfaz del tipo lápiz y papel, que permita interactuar con la aplicación mediante gestos con un lápiz y tomando notas sin usar teclado virtual o real.

4 **Problemáticas actuales**

La problemática del *levantamiento de procesos de negocios* esta siendo cada vez más relevante. Cada vez hay más empresas que buscan adoptar BPM como paradigma de descripción y documentación de procesos. Al mismo tiempo, la oferta de motores BPM y herramientas diseñadas para apoyar la construcción a sistemas orientados a BPM es

cada vez mayor, destacando algunos como **ORACLE BPM SUITE**, **BIZAGI SUITE** e **IBM BPM**. Por lo tanto, el abordar los problemas que siguen surgiendo en la tarea del levantamiento de proceso es no sólo interesante, sino además muy contingente.

El conocimiento está repartido entre los actores del proceso y los encargados de éstos, y los problemas de comunicación entre ambos y los analistas es una de las principales fuentes de problemas. Por ende la toma de datos en terreno y en forma colaborativa es una solución consistente con dichos problemas (Baloian, y otros, 2011).

Los escenarios y culturas organizacionales son diversos, por lo que tener la flexibilidad y soporte que una aplicación como esta ofrece, es fundamental para enfrentarse al levantamiento de procesos de negocios (Bhaumik & Rajagopalan, 2009).

El desarrollo de aplicaciones para móviles y en **HTML 5** es un desafío interesante por cómo se posicionan cada vez más como alternativas de vanguardia en el desarrollo de herramientas de software. Por otro lado, se resuelve al mismo tiempo la problemática de desarrollar una aplicación que sea independiente de la plataforma en que se utiliza.

5 Alcance y objetivos

5.1 *Objetivo general*

El objetivo de esta memoria es diseñar e implementar una aplicación basada en los principios de **NETSKETCHER**, que apoye la tarea de levantamiento de procesos mediante modelos **BPMN**, orientada principalmente al trabajo colaborativo y móvil. Esta aplicación permitirá el trabajo en terreno² de varios usuarios, simultáneamente o en forma asíncrona, mediante una interfaz simple que permitirá hacer del modelamiento **BPMN** una herramienta de apoyo a los procesos de levantamiento de procesos de negocio.

La aplicación a desarrollar se hará cargo de los objetivos y funcionalidades que han sido planteadas originalmente para **NETSKETCHER** y, mediante el uso de nuevas tecnologías, se le dará un nuevo enfoque que aumente su alcance, facilitando el cumplimiento de los objetivos y requerimientos propuestos. De esta manera se encuentra dentro del alcance de esta memoria el desarrollo de esta aplicación, y así dar una implementación concreta a los requerimientos y necesidades planteadas por (Baloian, y otros, 2011).

² Se entiende por *trabajo en terreno* al trabajo realizado por los analistas para el levantamiento de procesos, llevado a cabo en las instalaciones en las que los procesos son llevados a cabo y posiblemente en colaboración con quienes participen de dichos procesos.

NETSKETCHER está diseñado para ser usado por uno o más analistas, con una interfaz que permite trabajar en el modelo BPM durante una entrevista, sin entorpecer la misma, como si de tomar notas se tratase.

La nueva herramienta estará enfocada no solo a los analistas sino a cualquier involucrado en el proceso, con o sin conocimientos en **BPMN**, buscando no solo no entorpecer la entrevista, sino complementarla, haciéndose parte de la misma y facilitando el uso de otras estrategias para levantamiento de procesos.

5.2 Objetivos Específicos

El cumplimiento del objetivo general se lleva a cabo a través de los siguientes objetivos específicos:

- Entender el estado del arte en prácticas y herramientas para el modelamiento BPMN, de modo de asegurar que la aplicación propuesta cumpla con los estándares actuales y aporte en comparación a las herramientas existentes.
- Contar con una herramienta que apoye la construcción de modelos **BPMN** en equipos de dispositivos móviles como *smartphones* y *tablets*.
- Poder completar y complementar el trabajo hecho en dispositivos móviles en algún sistema más completo y cómodo en equipos de escritorio.
- Contar con una herramienta que apoye el trabajo en conjunto de analistas y actores de proceso en el levantamiento y modelamientos de éstos.
- Contar con mecanismos para poder compartir y coordinar el trabajo de modelamiento de procesos en **BPMN** entre distintos usuarios.

5.3 Resultado esperado

El resultado esperado al final del proyecto de memoria, es una aplicación *diseñada para dispositivos móviles*, en particular Smartphone y Tablets con **IOS** o **ANDROID**, cuya principal funcionalidad es la de *modelar procesos usando BPMN*.

La aplicación debe permitir el trabajo simultáneo de varios participantes en el mismo modelo de un proceso, y el persistir, compartir y administrar estos modelos entre varios participantes. Debe permitir la participación *tanto por analistas expertos como usuarios sin experiencia en BPMN* pero con conocimiento del proceso.

Por último, es deseable que la herramienta permita complementar el trabajo en equipos de escritorio, por ejemplo mediante la *exportación del trabajo realizado para mejorar y completar el desarrollo del modelo* en herramientas más completas, que muchas veces permiten utilizarlo directamente como punto de partida del desarrollo de sistemas

orientados a dar soporte al proceso, como **ORACLE BPM SUITE**, **IBM BPM**, **BIZAGI SUIT**, etc.

6 Estrategia del proyecto

Este proyecto de memoria consistió entonces principalmente en llevar a cabo la implementación concreta de una aplicación que satisfaga las características expuestas en los capítulos precedentes. En forma similar a **NETSKETCHER**, la aplicación desarrollada para esta memoria cuenta con los siguientes principios de diseño: *móvil*, *colaborativa*, y con una *interfaz simple y flexible pero completa*.

6.1 Aplicación móvil

Para desarrollar en *dispositivos móviles* se evaluaron las siguientes alternativas:

- Desarrollar específicamente para una plataforma. En particular **IOS** o **ANDROID**.
- El utilizar algún framework de desarrollo cross-plataform para **IOS** y **ANDROID**, que son los sistemas operativos móviles más utilizados en smartphone, como **UNITY**, **CORONA** o **THE PARTICLE SDK**.
- Desarrollar la aplicación como una aplicación web utilizando **HTML5**.

De las tres alternativas se escogió **HTML5** por las siguientes razones:

- Es un estándar independiente de la plataforma, por lo que, además de ser multi-plataforma, es robusto a cambios de versiones.
- Pese a que aún ningún navegador soporta toda la especificación de **HTML5**, el soporte actual está bastante avanzado y está equiparado entre los navegadores de escritorio y los móviles, al menos en los puntos más relevantes para este proyecto que son el uso del *canvas*, *SVG*,³ y comunicación.
- Salvo el contar con un navegador compatible, no se necesita instalar ningún otro software, por lo que hay menos trabas para que usuarios casuales puedan utilizar la herramienta.⁴
- Existen varios *frameworks* que facilitan el desarrollo en **HTML5**, algunos específicos para móviles como *JQuery Mobile* o *Titanium*.

³ Ver Anexo A. Características y Comparación entre Canvas y SVG, página 45.

⁴ Ver Anexo B. Compatibilidad de Browser con HTML5, página 46

- No es necesario aún hacer uso de funcionalidades nativas de los equipos móviles más allá de lo que **HTML5** permite.
- **HTML5** se perfila como una tecnología fuerte en desarrollo de aplicaciones a futuro, lo cual es muy importante si se tiene en cuenta la mantención y futuros trabajos que se puedan realizar.

6.2 *Aplicación colaborativa*

La aplicación desarrollada permite la colaboración entre los diversos usuarios en dos niveles:

- Trabajo simultáneo de varios usuarios sobre un mismo proceso, en distintos dispositivos a modo de *pizarra compartida*.
- El trabajo realizado se puede almacenar en línea de forma de poder ser compartido y/o retomado en un futuro por el o los mismos usuarios u otros nuevos que también tengan acceso a él.

6.3 *Interfaz simple y flexible, pero completa*

La interfaz de usuario y su usabilidad es la principal diferencia entre **NETSKETCHER** y la aplicación desarrollada. Existen dos motivos de fondo que motivan los cambios propuestos a la interfaz en comparación a **NETSKETCHER**:

- **HTML5** ofrece grandes ventajas en el tema de portabilidad de la aplicación que se desean explotar.
- Se quiere potenciar más algunas técnicas que proponen hacer participar a los actores y encargados del proceso en la elaboración de los modelos.

Para enfrentar estos temas, se propuso diseñar una interfaz con las siguientes características:

- **Portable**, para que pueda ser utilizado tanto en un equipo de escritorio como móvil con la menor cantidad de cambios posibles. En equipos de escritorio es difícil usar un enfoque tipo lápiz y papel.
- **Intuitiva**, para ser usado por usuarios sin conocimientos de la herramienta. Una interfaz por gestos si bien es cómoda para quien la conoce puede ser compleja de usar para quien no la conoce.
- **Flexible**, pues debe permitir que se utilice informalmente por gente que no maneja el estándar **BPMN**, en la elaboración de modelos que poco a poco se vayan completando.
- **Sencilla**, para poder ser utilizada como parte de la reunión sin entorpecerla.

- **Completa**, es decir, que permita hacer modelos que respeten la especificación **BPMN 2.0** (especificación actual).
- **Formal**, a diferencia **NETSKETCHER**, que tiene un aspecto de borrador informal. Debe tener un aspecto más formal de manera que los participantes sepan que están trabajando sobre el modelo y no sobre un bosquejo donde los *errores* u *omisiones* son aceptables.

La interfaz propuesta por la aplicación desarrollada fue basada en la que actualmente ofrece **BIZAGI PROCESS MODELER**, una de las herramientas de modelamiento **BPMN** mas utilizada en la actualidad para modelar pues se considera que cumple con todos los requerimientos y, además, esta ampliamente aceptada. Algunas de las características de **BIZAGI PROCESS MODELER** que motivaron tomarla como punto de partida son las siguientes

1. **Creación de elementos básicos en forma fácil (Drag & Drop)**. Ilustrado en la Figura 2.

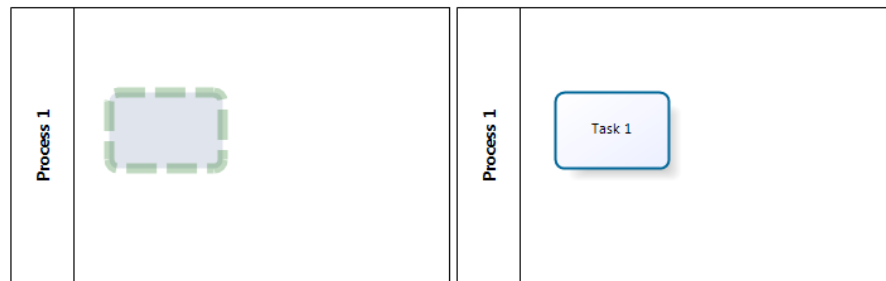


Figura 2. Característica de Drag & Drop.

2. **Edición con menú contextual**. Ilustrado en la Figura 3.

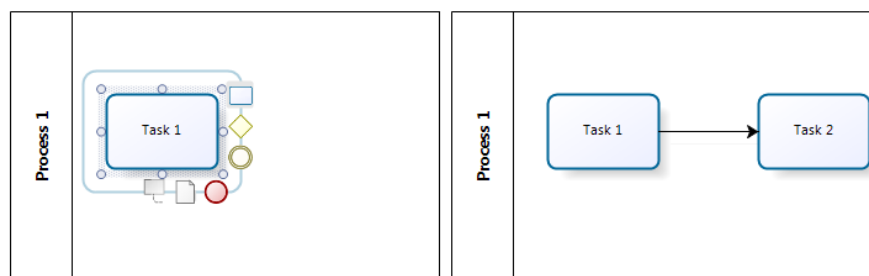


Figura 3. Edición con menú contextual.

3. **Inserción de nuevos artefactos**. Ilustrado en la Figura 4.

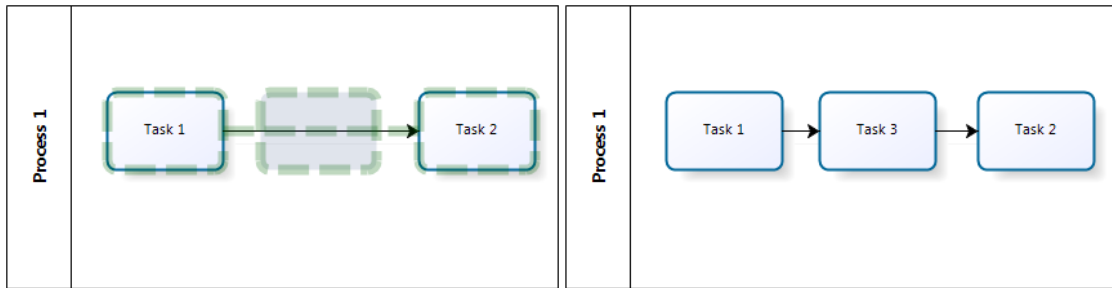


Figura 4. Inserción de nuevos artefactos.

4. **Flexibilidad gráfica.** Que permita salirse del estándar, como se ilustra en la Figura 6

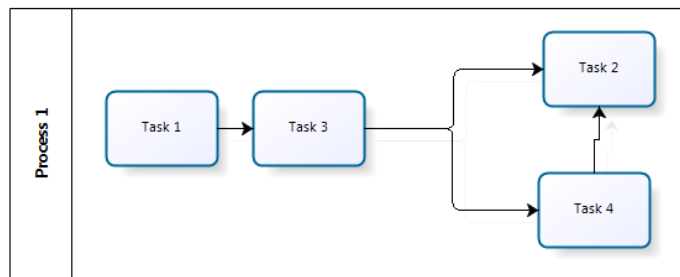


Figura 5. Flexibilidad respecto al estándar utilizado.

5. **Desarrollo evolutivo** hasta estar completos, como se ilustra en la Figura 6.

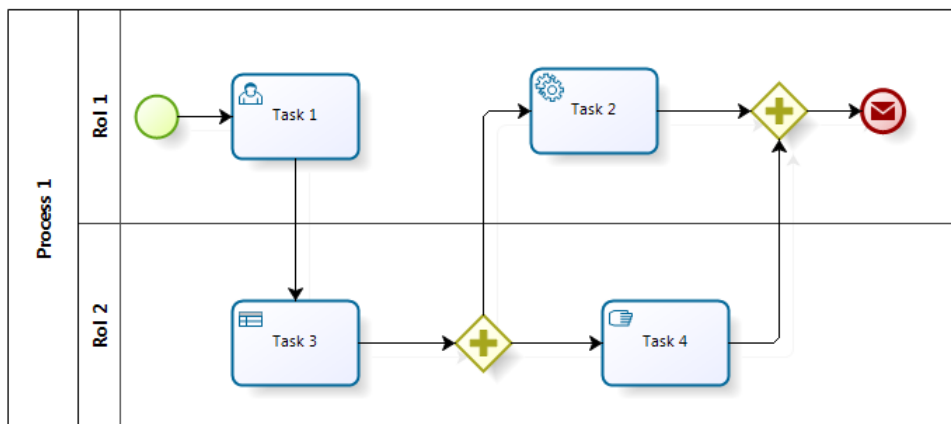


Figura 6. Desarrollo evolutivo.

Todas estas características fueron consideradas y son abordadas oportunamente a lo largo del desarrollo del presente documento.

Capítulo II

Mobiz: Aplicación BPMN

7 Introducción, alcance y objetivo

La primera etapa del proyecto consintió en la construcción de una aplicación monousuario para la elaboración de modelos **BPMN**. El resultado obtenido en esta etapa fue una aplicación web con dos características principales⁵:

1. Permite la construcción de modelos **BPMN** básicos mediante una interfaz orientada a dispositivos móviles.
2. Permite la construcción de modelos **BPMN** más completos mediante una interfaz orientada a navegadores de escritorio.

Por motivos de tiempo algunos artefactos del estándar **BPMN** fueron dejados fuera de la implementación para poder dar prioridad a la implementación de características más generales de interacción y herramientas de manipulación de los modelos. La razón de

⁵ Como esta aplicación será luego transformada en una aplicación distribuida, la persistencia y posterior acceso de modelos construidos no se incluyó en esta etapa del desarrollo sino en la etapa de construcción de la capa de comunicación.

esto fue el darle prioridad a los conceptos que se consideraron más relevantes, en cuanto a que permitan probar que la solución propuesta es una buena solución al problema planteado.

8 El modelador BPMN: Mobiz

MOBIZ es el nombre que se le dio a la aplicación construida durante este trabajo de título. En los capítulos anteriores se discutió lo útil que sería contar con una herramienta que permita por un lado el levantamiento de procesos en terreno en un esfuerzo conjunto entre analistas y agentes de las distintas partes de los procesos en estudio. En base a esto se postula que dicha herramienta tiene los siguientes requerimientos:

1. Una interfaz móvil que permita la construcción y edición de modelos en terrenos. Esta interfaz debe enfrentar las características y limitaciones de los dispositivos móviles de forma de ser suficientemente simple y eficiente para el levantamiento inicial (no tan detallado) de procesos en terreno.
2. Un mecanismo que permita continuar con el trabajo iniciado en los dispositivos móviles en alguna interfaz más completa en un equipo de escritorio. El trabajo en equipos de escritorio debe ser en alguna aplicación cuya interfaz esté a la altura de las aplicaciones actuales para la construcción de modelos BPMN.
3. El trabajo, al menos en dispositivos móviles, debe ser colaborativo, tanto en forma on-line, permitiendo que varios usuarios trabajen de forma simultánea en un modelo como en forma off-line, permitiendo que se compartan y distribuya el trabajo. Este requerimiento es enfrentado en el *Capítulo III: Framework de comunicaciones*.

9 Principios de Diseño

Para el desarrollo de **MOBIZ**, lo primero fue establecer una serie de principios de diseño sobre los cuales posteriormente se tomaron las decisiones más detalladas de la implementación. En esta sección se exponen y discuten dichos principios de modo de facilitar el entendimiento del diseño detallado.

9.1 HTML5

Se decidió que la implementación fuera realizada utilizando HTML5 pues esto ofrece una serie de ventajas, algunas relevantes para el desarrollo y otras que se reflejan en

características no funcionales de la aplicación solo por ser desarrollada con esta tecnología. Se destacan las siguientes características:

- Es un estándar independiente de la plataforma donde se utiliza. Por esto es de esperarse que su uso resulte en un menor esfuerzo al portar entre plataformas.
- Al ser una tecnología para aplicaciones web, no se requiere la instalación de software especial para utilizarla, sólo un navegador web compatible. Para el desarrollo de **MOBIZ** se consideró sólo **SAFARI** para dispositivos con **IOS** y **FIREFOX MOBILE** para dispositivos con **ANDROID**.
- No se requiere SDKs o plataformas especiales para desarrollar, a diferencia de otras tecnologías más específicas de los dispositivos. Esto simplifica considerablemente el proceso de desarrollo.

Sin embargo, esto implica algunas desventajas, como la dificultad de acceder a algunas características más específicas de los dispositivos o el no poder realizar procedimientos muy demandantes de recursos, pero ninguna de estas fueron relevantes para el desarrollo de **MOBIZ**.

9.2 Doble Interfaz

Para poder trabajar en los modelos en equipos de escritorio lo primero que se consideró fue en la exportación e importación modelos entre **MOBIZ** y alguna otra herramienta. Sin embargo, al estudiar más a fondo la tecnología y esbozar algunas opciones de solución, se decidió desarrollar una segunda interfaz para equipos de escritorio.

El enfoque para la construcción de estas interfaces fue el construir en primer lugar la aplicación con interfaz para equipos de escritorios, que debiera ser más completa, y luego su adaptación para dispositivos móviles. Las ventajas principales de abordar el problema de esta forma son las siguientes:

- Si se diseña bien la aplicación sólo hay tres temas de los que preocuparse al momento de hacer las adaptaciones para los equipos móviles: Las interacciones que involucran el uso del ratón/gestos, el tamaño de la ventana disponible para la aplicación y los problemas de compatibilidad que puedan suscitar de algunas funcionalidades **JAVASCRIPT** entre distintos navegadores. Todo esto, dado el uso de frameworks especializados, se considero que requeriría menos trabajo que la exportación e importación de modelos.
- Un computador portátil con navegadores de escritorio podría ser un dispositivo móvil válido para la toma del trabajo en terreno en algunas circunstancias, por lo que el contar con esta interfaz refuerza este requerimiento.
- Es mucho más cómodo contar con los modelos disponibles para ser accedidos directamente desde la aplicación para equipos de escritorios que tener que exportar e importar los modelos.

- Al ser la misma aplicación con distinta interfaz, se podrían utilizar las características colaborativas también entre equipos móviles y de escritorio. Además todas las características de la aplicación de escritorio podrían estar eventualmente disponibles en la aplicación móvil, sólo mediante la definición de la interfaz para accederlas.

10 Tecnologías a utilizar

Para el desarrollo de la aplicación se utilizaron las siguientes tecnologías:

- Para la implementación de toda la lógica por el lado del servidor se utilizó **JEE 7**. El servidor utilizado para desplegar la aplicación y disponibilizar los recursos web fue **TOMCAT 7**.
- La gran mayoría del desarrollo de **MOBIZ** fue hecho en el lado del cliente, con **HTML5**, en particular usando los frameworks **JQUERY** y **JQUERY MOBILE**. Este último fue especialmente útil pues, además de ofrecer una base sólida para el desarrollo de aplicaciones móviles, facilita la portabilidad de aplicaciones **HTML5** desde escritorio a dispositivos móviles. **JQUERY MOBILE** logra esto mediante la estandarización de algunos mecanismos que son exclusivos de dispositivos móviles para que puedan ser utilizados en navegadores de escritorio y viceversa.

11 Diseño de la solución.

En esta sección se expone el diseño de la solución implementada. Se intenta dar una visión completa de como esta construido, partiendo desde los conceptos más básicos y generales y terminando con los detalles más importantes de su implementación.

11.1 *Diseño general*

En las aplicaciones web tradicionales, la lógica del negocio subyacente se encuentra normalmente en el servidor de aplicaciones. Sólo el despliegue del contenido, y algunas interacción con el usuario se hacen utilizando **JAVASCRIPT** y **HTML/CSS**. Sin embargo, en este caso, y en muchas aplicaciones **HTML5** actuales con interacciones e interfaces de alto nivel, gran parte de la lógica de la aplicación va en la interacción con el mismo,

y puede ser implementado gracias a los conceptos introducidos por **HTML5** mediante el uso de **JAVASCRIPT** y **HTML/CSS**.

Si se quiere tener una aplicación fácil de adaptar para distintos dispositivos es muy importante que artefactos que implementen la lógica de la aplicación estén lo más desacoplados posible de la visualización misma. Esto se debe a que esta última es mucho más sensible a cambios entre plataformas, y si se logran separar bien, toda la implementación de la lógica debiera poder ser reutilizada. Para lograr esto, **MOBIZ** está construido siguiendo un modelo de capas. El diseño de **MOBIZ** considera dos capas, de presentación y de lógica de control, y un tercer tipo de artefactos cuyo objetivo es el de implementar la comunicación o integración entre estas capas.

11.1.1 Capa de Presentación

Los artefactos de esta capa tienen un doble propósito. Por un lado son responsables de desplegar al usuario la información que le entregue la capa lógica. Por otro lado reciben las entradas del usuario y se los entregan a la capa lógica.

Si bien en esta capa se utilizan algunos elementos básicos de interfaz, como botones y menús, el problema principal y más interesante es el del despliegue del modelo BPMN y la interacción con éste.

HTML5 ofrece dos mecanismos especiales para el despliegue y manipulación de dibujos, diagramas o cualquier otro tipo de elementos gráfico complejo que son **Canvas** y **svg** (más detalles en apéndice Anexo A1, página 72). Para **MOBIZ** se escogió utilizar **svg** como el mecanismo para desplegar el diagrama, siendo la principal razón el manejo de las interacciones con el usuario.

JAVASCRIPT incluye un mecanismo bastante completo para detectar y manejar distintos eventos causados por la interacción entre el usuario y los distintos elementos de la página. La base de este mecanismo es el poder detectar las interacciones entre el usuario y algún nodo particular del DOM de la página, siendo así el nodo la unidad mínima de control, es decir que no se puede de forma nativa interactuar con partes de un nodo, salvo que estas sean otros nodos hijos. Por otro lado, un **Canvas**, si bien puede ser manipulado incluso a un nivel de pixel por pixel, para efectos de interacción es un solo nodo del **DOM**. De esta forma, para interacciones complejas con un **Canvas** se deben construir mecanismos personalizados que, además de ser complejos de implementar, podrían ser costosos de realizar. Por el contrario, un diagrama en **SVG** se construye en base a la definición en **XML** de sus partes, por lo que cada parte del modelo es tratada como un nodo del **DOM** independiente, pudiendo así utilizarse los mecanismos predeterminados para el manejo de la interacción con el usuario.

11.1.2 Capa de Lógica de Control

Esta capa, como su nombre lo indica, esta compuesta por objetos lógicos que representan los componentes del modelo e implementan la lógica de control de las interacciones con el usuario. El Diagrama 1 muestra la estructura básica de las clases de esta capa.

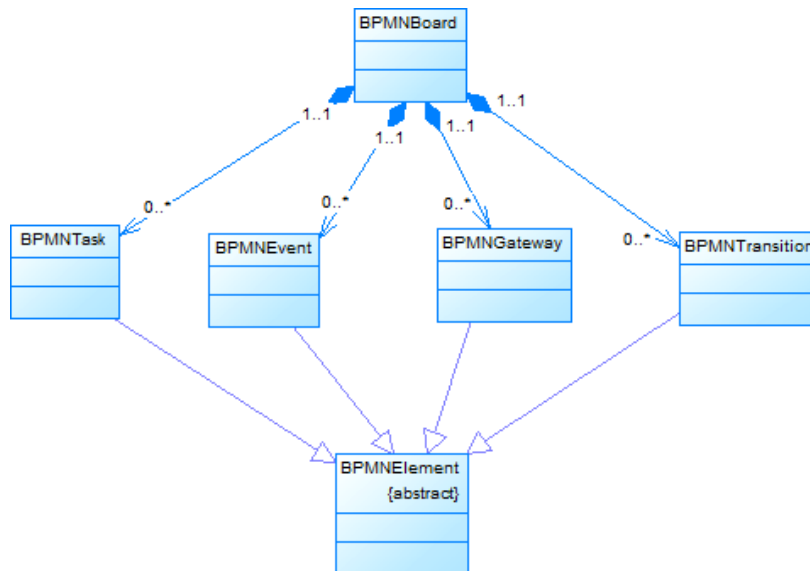


Diagrama 1: Clases Capa de Lógica de Control

La composición de cada clase es muy similar entre si. A grandes rasgos están compuestos de la siguiente forma:

- Una serie de atributos junto con los métodos para acceder y modificar sus valores.
- Una serie de funciones para el manejo de eventos provocados por interacciones con el usuario.
- Un constructor que, entre otras cosas, utiliza los artefactos de integración para relacionarse a nodos del **DOM** específicos y registrarse como su interfaz de control de eventos.

11.1.3 Artefactos de Integración

Los artefactos de integración son algunos artefactos de software que buscan abstraer lo más posible la comunicación entre las capas. Como se puede entrever de las subsecciones anteriores, estos artefactos deben encargarse de la comunicación en ambas direcciones entre las capas. Para la comunicación desde la capa de lógica de control a la de presentación, se necesita poder generar las alteraciones necesarias al **DOM** para desplegar la información correctamente. Para la comunicación entre la capa

de presentación y la de lógica de control se requieren mecanismo para asociar eventos lanzados por nodos del **DOM** a las funciones designadas para manejarlos.

Además, era necesario un mecanismo para acceder al objeto lógico que representa a un nodo en particular y viceversa. Para cada una de estas necesidades se utilizó un tipo de artefacto distinto. Estos artefactos son:

SVGManager

La clase **SVGManager** implementa todas las funcionalidades que se utilizaron para crear y manipular modelos usando **SVG**. Su construcción se basa en encapsular las funcionalidades de bajo nivel de **JAVASCRIPT** y **JQUERY** para la manipulación del **DOM**, en funcionalidades de más alto nivel, como la creación de figuras simples y complejas, aplicación de transformaciones y manejo de filtros y efectos.

Esta herramienta proveyó efectivamente un mecanismo para que los objetos lógicos pudieran manipular los nodos del **DOM** que fueran necesarios para el despliegue de la información, sin entrar en detalles de como están estructurados.

Eventos de jQuery

El framework **JQUERY** ofrece un mecanismo muy robusto y potente para el manejo de eventos. A grandes rasgos, este mecanismo permite asignar objetos y sus funciones como manejadores de eventos lanzados en el **DOM**. Además permite lanzar eventos e incluso definir nuevos eventos. Usando este mecanismo se logra abstraer efectivamente la generación del manejo de éstas.

La ventaja de utilizar este mecanismo queda especialmente clara cuando se cambia de un dispositivo que utiliza un ratón o algo equivalente, a un dispositivo táctil, donde los eventos no son los mismos. En dicho caso es muy útil el poder asignar el mismo manejador a distintas funciones o el crear eventos virtuales que encapsulen más de un tipo de evento real simultáneamente.

El objeto jQuery y sus Plugins

El framework **JQUERY** está basado principalmente en la utilización de una herramienta llamada **OBJETO JQUERY**. La principal característica que tiene este objeto es que cada instancia esta siempre ligada a un único nodo del **DOM**. Más específicamente cada objeto **JQUERY** representa un nodo del **DOM** y ofrece una interfaz de mayor nivel para manipularlo. Además, cabe indicar que es posible obtener la instancia (o en su defecto crear una nueva instancia) del **OBJETO JQUERY** que representa un nodo sólo conociendo el nodo y viceversa.

Una de las principales características del **OBJETO JQUERY** es que es extensible mediante la utilización de *plugins*. Lo que hace un plugin es básicamente agregar o reemplazar

algún método del **OBJETO JQUERY**. En este caso, se utilizó la siguiente estrategia para el diseño de los plugins:

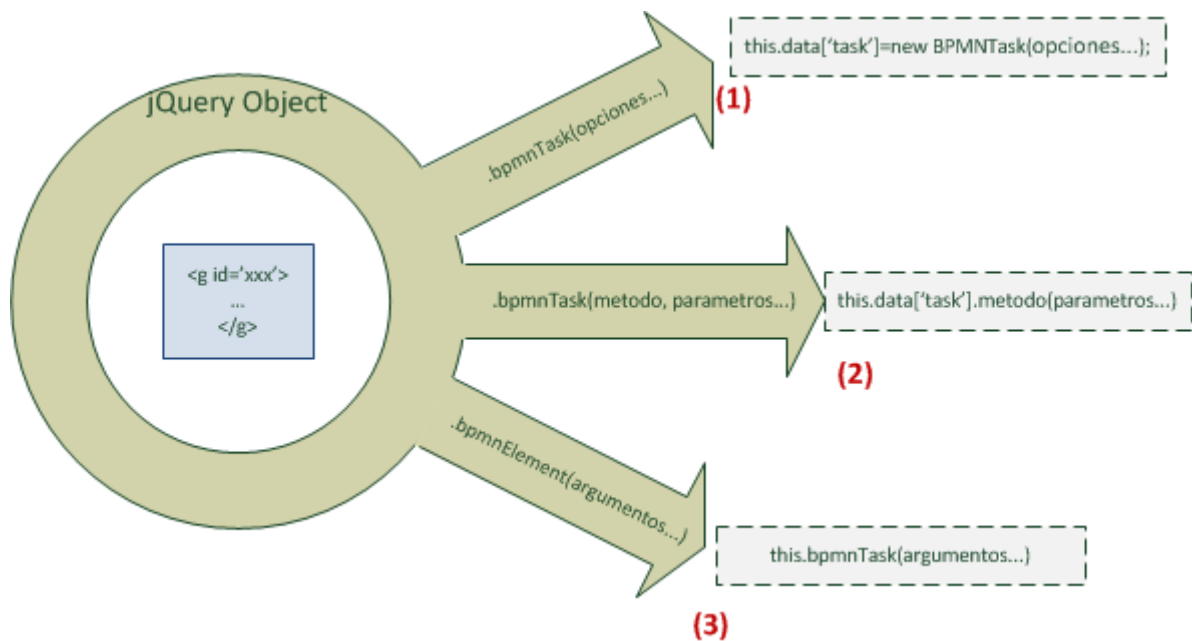


Figura 7. Definición de Plugins jQuery.

1. Se creó un plugin distinto por cada una de las 5 clases de la capa de lógica de control que se muestran en el Diagrama 1. Cada plugin crea una sola función con el mismo nombre que la clase a la que representa.
2. Salvo el método `bpmnBoard`, los métodos de los plugins solo pueden ser aplicados a objetos **JQUERY** que representen un nodo `<g/>` con el atributo `id` definido. Un nodo `<g/>` es simplemente un agrupador en diagramas **SVG** y que permite que se le apliquen transformaciones, como escalamientos o rotaciones, que afectaran a todos los nodos que contenga.
3. Al inicializar un plugin, se crea una nueva instancia de la clase a la que representa y ésta se registra en el **OBJETO JQUERY** (1).
4. Cuando la llamada al método del plugin tiene como primer argumento un `string`, esto se traduce en una llamada a un método del objeto que lo representa (2). De lo contrario se considera que se desea inicializar el plugin.
5. El plugin `bpmnElement` no realiza ninguna inicialización. Su funcionamiento consiste en detectar cual de los otros plugins está inicializado para el **OBJETO JQUERY** en que se invoca, y delega la ejecución de lo solicitado al plugin correspondiente (3).

11.2 Diseño de interfaces

Como se mencionó anteriormente, el diseño de las interfaces se realizó en dos etapas. La primera fue el diseño de la interfaz para navegadores de escritorio, pues ésta debía contener todas las funcionalidades disponibles. Luego, mediante la adaptación de la primera, se diseñó la interfaz para dispositivos móviles.

11.2.1 Interfaz para navegadores de escritorio

Esta versión de la interfaz fue basada en la que propone BIZAGI en su aplicación de escritorio BIZAGI PROCESS MODELER, sólo con algunos ajustes y simplificaciones para adaptarse mejor a la tecnología utilizada y el tiempo disponible.

A continuación se mencionan las principales características y funcionalidades ofrecidas por esta interfaz.

Representación de los artefactos

La representación gráfica de los artefactos del estándar BPMN se hizo siguiendo la definición oficial de BPMN 2.0 (Business Process Model and Notation (BPMN) Version 2.0, 2011) y las representaciones usadas por BIZAGI PROCESS MODELER. Las representaciones de cada uno de los artefactos incluidos actualmente en MOBIZ son las que se muestran en la Figura 9.



Figura 8 Artefactos del Modelo.

En la ilustración se muestra que se incluyeron cinco *clases* de artefactos cada uno con una forma y color característicos. Además cada artefacto puede ser especificado mediante la asignación de un *tipo*, que se representa como un icono adicional.

Barra de artefactos

La primera parte de la interfaz con la que un usuario debe interactuar es la barra de artefactos.



Figura 9. Barra de Artefactos.

Esta barra de herramientas permite la creación de cada uno de las distintas clases de artefactos BPMN al arrastrar alguno de los iconos dentro del espacio dispuesto para la construcción del modelo.

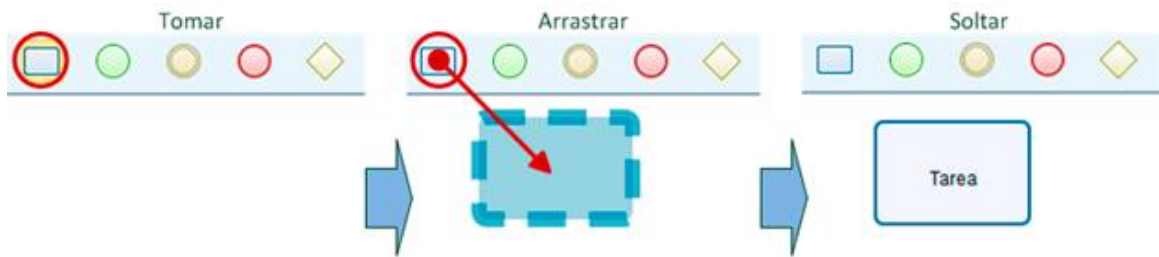


Figura 10. Drag & Drop de Artefactos Nuevos.

Como se puede apreciar de la Figura 10, esta sola acción crea un artefacto completamente definido, pues se crea con un valor por defecto para cada uno de sus atributos, incluyendo el tipo de artefacto y su nombre.

Otra característica de la barra de artefactos es que permite la creación de artefactos intermedios, es decir en medio de una transición entre dos artefactos, mediante su colocación sobre la transición que se desea intervenir. Esto se ilustra más claramente en la Figura 11:

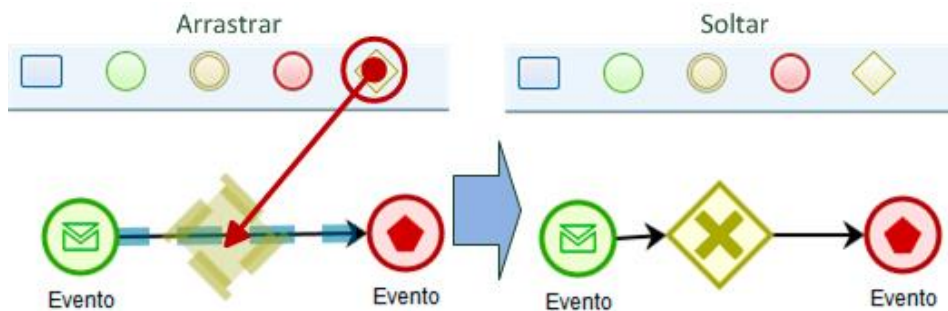


Figura 11. Creación de artefactos intermedios.

Interacción con el usuario usando un ratón

Con la intención de indicar al usuario cómo y con qué artefactos puede interactuar usando el ratón, se utilizaron respuestas gráficas a la acción de pasar el puntero sobre un artefacto (*MouseOver*), para indicar que se puede interactuar con el, y seleccionar un artefacto. Esto último se puede referir a una selección explícita al hacer clic sobre el artefacto, o de forma implícita, para indicar que alguna acción en curso, sea cual sea, afectará al artefacto resaltado. La Figura 12 muestra como se destaca un artefacto en ambos casos.

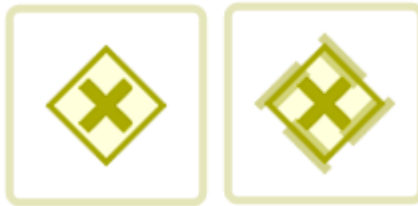


Figura 12. MouseOver (izquierda) y selección de artefactos (derecha).

Ajuste automático de texto

Uno de los cambios de la interfaz de **MOBIZ** con respecto a la interfaz en la que se basa, es el ajuste automático del texto dentro de un recuadro determinado. El resultado de esta funcionalidad es que al cambiar el tamaño del texto contenido, por ejemplo, en una tarea, el tamaño del cuadro que la representa se ajusta automáticamente, teniendo en cuenta el mantener una buena proporción entre el alto y el ancho del cuadro. Un ejemplo del resultado del ajuste de texto automático se puede ver en la Figura 13.

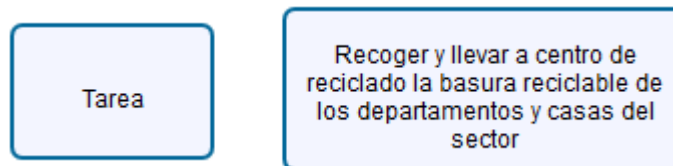


Figura 13. Ajuste automático de tamaño de contenedores según texto.

Menú contextual

Una de las más interesantes características de Mobiz es el uso de un menú contextual al intentar interactuar con algún artefacto. Este menú, no despliega acciones a realizar sobre el artefacto seleccionado, sino que despliega una lista de posibles tipos de artefactos hacia los que se puede crear una transición.

Este *menú contextual* específicamente ofrece dos alternativas de uso. La primera es la creación de un nuevo artefacto que quedará inmediatamente ligado al objeto original mediante una transición. Esto, como se puede ver en la Figura 14, se hace arrastrando el icono del artefacto dentro del menú contextual a la posición deseada para el nuevo artefacto, de forma similar a como se haría usando la barra de artefactos.

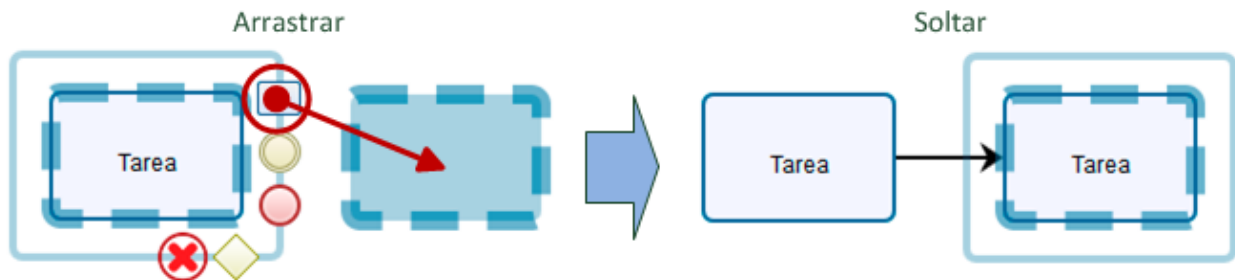


Figura 14. Creación de artefactos relacionados desde menú contextual.

La otra alternativa es el crear una transición entre dos artefactos ya creados. Esto se hace de la misma forma como se haría para crear un nuevo artefacto, pero en lugar de arrastrar el icono a un espacio en blanco, se arrastra sobre el artefacto de destino. Esto se puede ver claramente en la Figura 15. Intencionalmente se incorporó la restricción de que esto no funciona si el icono arrastrado no coincide con el tipo de artefacto sobre el que se arrastra, caso en el cual sólo se crea un nuevo artefacto⁶. Esto se diseñó de esta forma para evitar posible ambigüedades en lo que el usuario desea hacer.

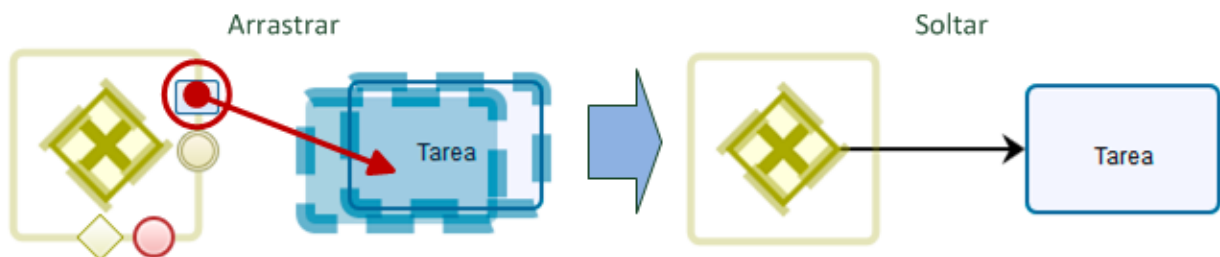


Figura 15. Enlazar dos elementos desde menú contextual.

⁶ Notar que el artefacto de destino es marcado como seleccionado sólo cuando se arrastra el icono correcto y la creación de la transición es posible.

Otras características

Además de las características anteriormente mencionadas se incluyeron algunas otras características más sutiles u obvias, desde el punto de vista de la usabilidad, pero que vale la pena destacar, pues su detección e implementación no resultaron sencillas. Estas características son:

- **Tamaño automático del área de trabajo:** El tamaño del lienzo en el que se construye el modelo, en lugar de partir con un tamaño muy grande, se va ajustando al tamaño del modelo. Esto evita algunos problemas producto del exceso de espacio. Un ejemplo de esto es cuando un modelo no es construido en alguna esquina del lienzo y después cuesta trabajo encontrarlo.
- **Drag & Drop de artefactos:** Todos los artefactos del modelo son desplazables mediante *Drag & Drop* para cambiar su posición. Esto incluye el ajuste automático de cualquier transición hacia o desde el artefacto desplazado.
- **Zoom:** Se incluyó un ajustador de zoom para poder ajustar el tamaño del modelo relativo a la pantalla. Este efecto se logra especialmente bien gracias al uso de SVG para la creación del modelo, pues, como el nombre lo indica, SVG es directamente escalable sin pérdida de calidad. Este zoom se diferencia del que ofrecen los navegadores en que solo afecta al modelo y no al resto de la interfaz.
- **Teclas de acceso rápido:** Se incluyeron algunas teclas de acceso rápido para las funcionalidades de eliminación de artefactos (tecla suprimir), desplazamiento pixel a pixel de artefactos (flechas del teclado) y zoom (tecla shift + rueda del ratón).
- **Doble clic en artefactos:** Al hacer doble clic en algún artefacto (incluyendo transiciones) se despliega un pop-up con un formulario que contiene toda la información del artefacto seleccionado y permite editar el nombre, la descripción y/o tipo, esta última opción está disponible dependiendo de la clase de artefacto de la que se trate.

11.2.2 Interfaz para navegadores móviles

La interfaz para navegadores móviles se construyó mediante la modificación de la interfaz ya construida para navegadores de escritorio. Por esta razón, en esta sección, para describir la interfaz se utilizará un enfoque basado en describir principalmente las diferencias entre las interfaces más que en describir las características de ésta.

Elementos de interacción externos al diagrama

Uno de los principales desafíos en la adaptación de la interfaz, especialmente para smartphones, fue lograr optimizar el uso del poco espacio con el que se dispone en la pantalla.

El primer paso para hacerlo fue el hacer que el diagrama utilice el 100% del espacio disponible en la pantalla. Para lograrlo se utilizó una estrategia en la que todos los objetos de interacción externos al modelo mismo se incluyeron en las secciones de encabezado y pie de la página, las cuales pueden ser ocultadas o traídas a la vista al tocar la pantalla. El resultado de esto se muestra en la Figura 16.

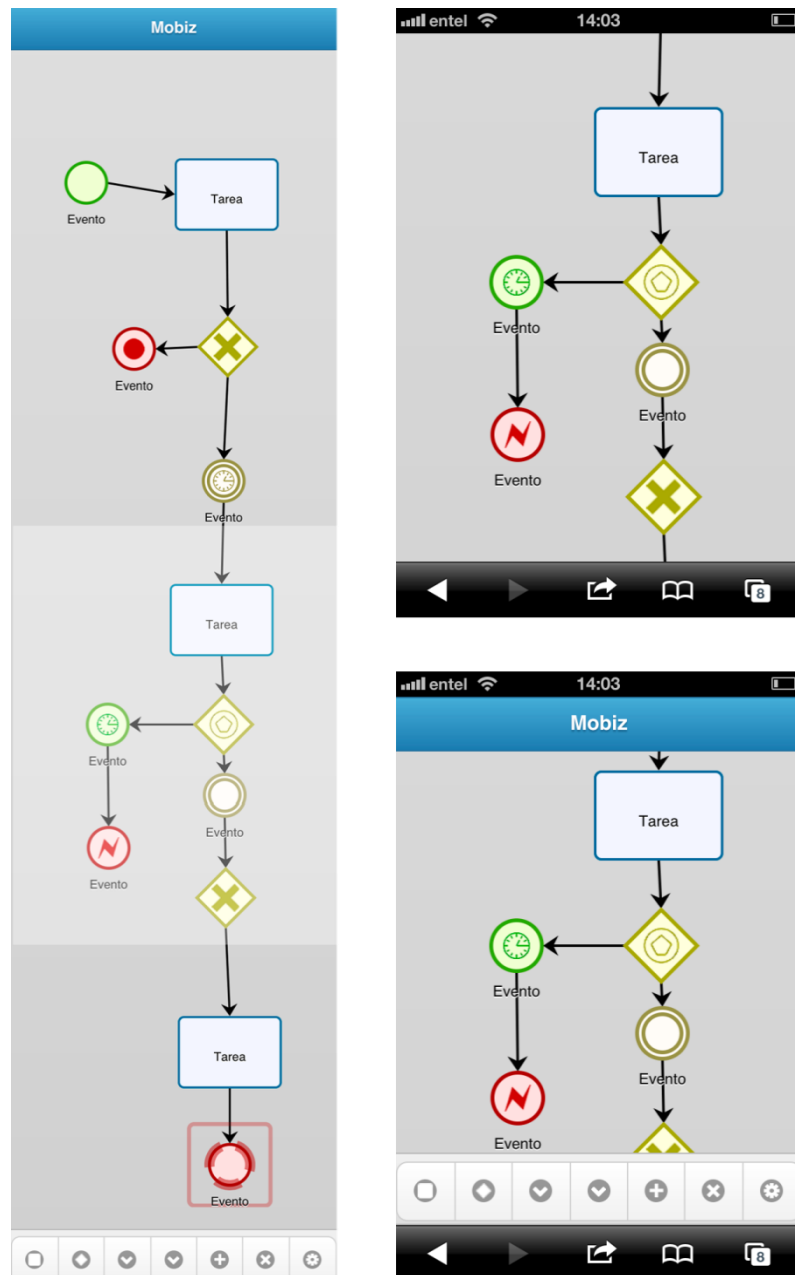


Figura 16. Encabezado y pie de página interfaz móvil.

La Figura 16 muestra a la izquierda una vista completa de la estructura real de la página y a la derecha muestra capturas de pantalla de como se visualiza una sección intermedia de la pagina con el encabezado y pie de pagina ocultos (arriba) y a la vista (abajo).

Elementos de la interfaz

En esta subsección se mostrarán y describirán las distintas partes que componen la interfaz móvil de **MOBIZ** y sus objetivos. El detalle de como se interactúa con estos componentes es descrito en la siguientes subsecciones de *Interacción con la interfaz* y de *Navegación*.

La interfaz móvil de **MOBIZ** esta compuesta por tres partes principales que son un encabezado, el cuerpo de la página y un pie de página, como se aprecia en la Figura 17.

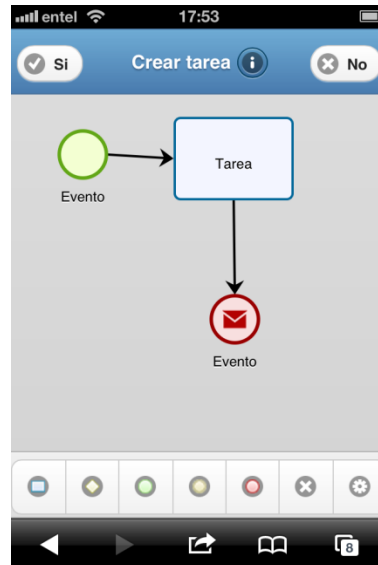


Figura 17. Interfaz móvil de Mobiz.




El encabezado, mostrado en la Figura 18, tiene por objetivo el mostrar información sobre la acción en curso. Esto se hace cambiando el texto que despliega a una descripción breve y simple de la acción que se esta realizando (en el ejemplo de la imagen es la creación de una nueva tarea). Si la acción tiene información adicional esta se despliega en una ventana emergente al tocar el botón .



Figura 18. Encabezado interfaz móvil.

Además, cada acción debe ser confirmada o cancelada, por lo que despliega la opción de hacer esto, mediante los botones  Si y  No respectivamente.

El *pie de página* se utiliza para desplegar el menú de herramientas. Este menú busca remplazar tanto a la barra de herramientas como al menú contextual que se usan en la interfaz para escritorio. Pretende poder realizar las mismas tareas sin tener que usar un ratón para arrastrar elementos. Además, esta barra contiene un botón adicional para la edición de la información de los artefactos, puesto que en la interfaz para escritorios esto se hacía haciendo *dobles clic*, lo que en la interfaz móvil está reservado para otra funcionalidad. El menú completo se puede ver en la Figura 19.

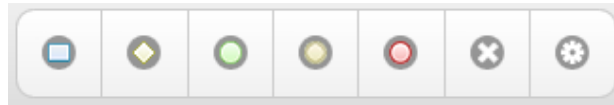


Figura 19. Menú interfaz móvil.

Finalmente está el cuerpo de la página en donde se despliega el diagrama y se puede interactuar con él. Esta sección utiliza todo el espacio disponible, salvo el utilizado por el encabezado y el pie de página cuando estos están a la vista.

Navegación

Se entiende por navegar a través del diagrama a la acción de modificar el punto de vista desde el que se observa. En el caso de **MOBIZ**, como es el habitual en aplicaciones móviles, esto se hace mediante gestos. Los gestos básicos que se utilizan se describen en la Figura 20.



Figura 20. Gestos básicos para dispositivos móviles.

- **Acercar y Alejar:** Se utiliza para manejar el zoom del modelo con respecto a la pantalla. Al hacer el gesto de acercar se verán los artefactos más grandes y cabrá una porción menor del diagrama en la pantalla. Si se hace el gesto de alejar los artefactos se verán más pequeños y cabrá una porción mayor del diagrama en la pantalla.

- **Arrastrar:** Esto cambiará la posición relativa del diagrama con respecto a la pantalla en una magnitud proporcional a la distancia que se arrastre el dedo sobre la pantalla. Esto provoca que se muestren partes del diagrama que no se podían ver anteriormente pero ocultando otras que estaban a la vista.
- **Barrer:** El efecto es similar al de arrastrar, con la excepción de que la magnitud del desplazamiento del diagrama es proporcional a la velocidad con la que se hizo el gesto y no a su longitud.
- **Tocar dos veces sobre un artefacto:** Esto, además de seleccionar el artefacto, hará zoom de forma que éste ocupe la mayor parte de la pantalla y desplazará el diagrama de forma que el artefacto seleccionado quede al centro de la pantalla.
- **Tocar dos veces en el fondo:** Esto alternara entre dos niveles de zoom, uno donde los artefactos se ven muy grandes y otro donde se ve gran parte del diagrama simultáneamente. Además desplazará inmediatamente el diagrama de forma que el punto en el que se tocó quede al centro de la pantalla.
- **Barrer horizontalmente con dos dedos:** Este es un gesto especial que solo se puede hacer cuando hay algún elemento seleccionado. El objetivo de este gesto es poder navegar a través de las transiciones entre los artefactos **BPMN**, en otras palabras, siguiendo las flechas del diagrama de a una transición por gesto. De esta forma, al hacer el gesto hacia la derecha se seguirá alguna flecha (al azar si hay más de una) y se percibirá un efecto similar al tocar dos veces en el artefacto al que esta flecha apunte. Lo mismo sucede si se hace el gesto hacia la izquierda, pero se seguirá alguna flecha en forma inversa. Si no existe ninguna flecha en la dirección seleccionada nada sucede.
- **Barrer verticalmente con dos dedos:** Al igual que el gesto anterior, este también se utiliza para recorrer el diagrama de a una transición por gesto. Como se explicó previamente, el gesto anterior escoge al azar alguna flecha para recorrer y selecciona el elemento de origen o destino de esta flecha. Barrer verticalmente con dos dedos, si se realiza gusto después del gesto anterior, permite ir rotando entre las diferentes alternativas que se podrían haber escogido para la navegación al hacer el barrido horizontal.

Interacción con la interfaz

Para la interfaz de escritorio se hizo intensivo uso del arrastre de elementos para la interacción con el usuario. Este tipo de interacción, si bien puede emularse en dispositivos móviles, en el caso de **MOBIZ** los gestos que podrían utilizarse para hacerlo están reservados para la navegación. Es por esto que se diseñó un mecanismo de interacción por pasos de la siguiente forma:

1. Se selecciona un artefacto o una posición vacía dentro del diagrama. Esto define el contexto.

2. Se toca un botón en el menú en el pie de página. Esto, según el contexto, definirá alguna acción que será desplegada en el encabezado de la página.⁷⁸
3. Opcionalmente se puede seleccionar un segundo elemento o posición en el diagrama. Esto modificara la acción a realizar. No todas las combinaciones entre elementos y/o posiciones son validas para todos los tipos de acciones.⁹
4. Si el o los elementos seleccionados son validos para la acción en curso se debe aceptar o cancelar la acción tocando el botón correspondiente en el encabezado. De aceptarse la acción se realiza, de cancelarse se vuelve al paso 1. Si la acción no es válida, el botón para aceptar no se despliega.

En la siguiente tabla se exponen las acciones posibles.

| Contexto | Acción | Elemento 2 | Resultado |
|---------------------|-------------------------|-------------------|---|
| (Elemento 1) | | | |
| Transición | Crear artefacto clase X | Ninguno | Crea un artefacto de clase X en medio de la transición |
| Artefacto | Crear artefacto clase X | Ninguno | Crea un artefacto de clase X con una transición desde el artefacto seleccionado hacia el nuevo, en una posición arbitraria, cercana al artefacto de origen. |
| Artefacto | Crear artefacto clase X | Artefacto clase X | Crea una transición desde el elemento 1 hasta el elemento 2 |
| Artefacto | Crear artefacto clase X | Espacio en blanco | Crea un artefacto de clase X con una transición desde el artefacto seleccionado hacia el nuevo en la posición del espacio seleccionado. |
| Artefacto | Crear artefacto clase X | Transición | Crea un artefacto de clase X con una transición desde el artefacto seleccionado hacia el nuevo que además esta en medio de la transición seleccionada. |

⁷ Si la acción escogida es la ultima (configurar), en el segundo paso termina la interacción pues se despliega inmediatamente una ventana para la edición de la información.

⁸ Existe una acción especial que no esta en el menú que es la de mover un elemento. Para activarla, en lugar de seleccionarla en un menú se debe tocar y mantener sobre el elemento.

⁹ Si se escogió la acción de eliminar se bloquea la navegación y selección de artefactos por lo que el paso 3 se omite siempre.

| | | | | |
|----------------|-------|-------------------------|---|---|
| Artefacto | Mover | Espacio en blanco | Mueve el artefacto a la posición del espacio seleccionado | |
| Artefacto | Mover | Transición | Coloca el artefacto en medio de la transición seleccionada. | |
| Espacio blanco | en | Crear artefacto clase X | Ninguno | Crea un artefacto de clase X en la posición del espacio seleccionado. |

11.3 Detalles de la implementación

11.3.1 Grupos y Capas

Una de las principales ventajas cuando se utiliza *SVG* en una aplicación web es que los elementos del dibujo *SVG* son también tratados como elementos del *DOM* y pueden ser manejados en forma similar, incluyendo el uso de clases y hojas de estilo. Sin embargo, no todos las propiedades y estilos de un elemento *HTML* común sirven para elementos *SVG* y viceversa. En particular, durante la implementación de *MOBIZ* se encontraron dos casos especialmente relevantes.

La primera diferencia importante surgió en el manejo del posicionamiento y tamaño de los elementos. A diferencia de con los elementos *HTML* tradicionales, es imposible manejar la posición o tamaño de los elementos *SVG* mediante estilos, y lo que es más, las propiedades para hacerlo directamente en el elemento (sin usar estilos) difieren de elemento en elemento. Esto produce serios problema en el manejo de posiciones y tamaños de elementos arbitrarios pues se debe detectar primero que tipo de elemento es y, según el caso, utilizar las propiedades correspondientes.

La segunda diferencia importante sucede en el manejo del orden de profundidad de los elementos superpuestos. Cuando dos elementos *HTML* están superpuestos, se puede definir cual de ellos irá más arriba usando la propiedad *z-index*. En *SVG* no existe ninguna manera de hacerlo y el elemento que va más arriba siempre es el que está definido después. Esto implica que hay que tener especial cuidado en donde se colocan los nuevos elementos para que se respete el orden deseado, por lo que es muy importante tener contenedores o marcas para poder encontrar en forma eficiente la posición correcta dentro del *DOM*.

Para poder manejar de buena forma estas situaciones, se uso el elemento de agrupación *<g/>* de *SVG*. Este elemento tiene dos características principales.

- Puede ser usado como solo un contenedor que no genera ningún efecto visible en el dibujo. Esto lo convierte en un elemento de marca ideal para poder definir secciones dentro del dibujo *SVG*.

- Se les puede aplicar transformaciones que afectan transitivamente a todos los elementos que contenga. En particular permite escalarlo y trasladarlo, dando así no sólo una manera de estandarizar este proceso independiente del contenido, sino que permite aplicarlo a varios elementos a la vez, lo cual es ideal si se considera que cada artefacto BPMN está compuesto por varios elementos SVG.

Luego, usando estos agrupadores se definió una estructura del diagrama que consiste en un grupo principal, en el que se maneja la posición y zoom del modelo completo y que contiene tres grupos, que representan tres capas con distinto nivel de profundidad.

Finalmente, cada artefacto consistirá, en a lo más, un grupo por cada capa, en donde se agrupen los elementos que lo componen. La Figura 21 grafica lo anteriormente mencionado e indica el uso que se le dio a cada una de las capas.

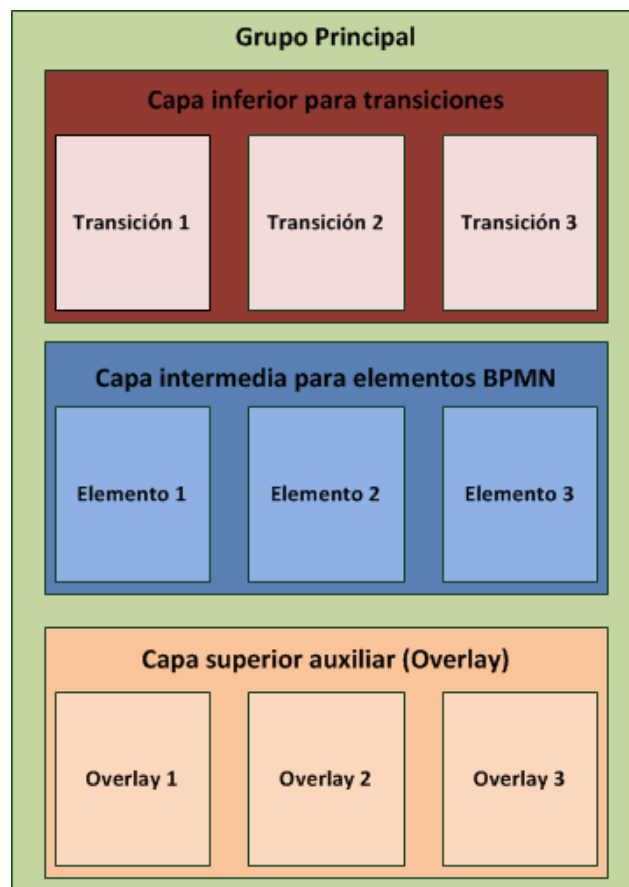


Figura 21. Estructura de grupos y capas de Mobiz.

11.3.2 Interacciones entre elementos

Algunas de las características que se construyeron implicaban que un elemento debía reaccionar cuando otro elemento se colocaba sobre él. La forma que parece más lógica

para implementar esto, es manejando el evento en los elementos fijos onmouseover mientras se arrastra algún elemento. El problema para implementar esto es que los eventos del ratón son solo detectados por el primer elemento que los recibe, en este caso el elemento que se está arrastrando, y nunca llegan al elemento de más abajo.

Para poder solucionar este problema se implementó una capa superior llamada *Overlay* (Figura 21). En esta capa el elemento fijo tiene un elemento auxiliar transparente que tiene la misma forma y posición y cuya función es delegar el manejo de cualquier evento detectado al elemento original que representa. De esta forma, pese a que el elemento que se arrastra se vea gráficamente sobre cualquier otro elemento del diagrama, éste sigue estando en la capa intermedia y, por ende, por debajo de cualquier elemento del *Overlay*. Esto permite que el evento sea detectado y manejado correctamente.

11.3.3 Drag & Drop

Las librerías existentes en **JAVASCRIPT** para el manejo de *Drag & Drop* están normalmente implementadas usando las propiedades de estilo para el manejo de los elementos. Sin embargo, como se mencionó anteriormente en esta sección, los elementos de *SVG* no soportan estas propiedades. Es por eso que se implementó una nueva librería que utiliza la propiedad de transformación de los grupos de *SVG* para trasladar los artefactos de una posición a otra. Esto además aprovecha la forma en que está construido el *SVG* del modelo, en donde las distintas partes de la representación de un artefacto están contenidas en un mismo grupo, de modo de poder trasladarlo como si fuera una sola unidad.

Sin embargo, para lograr esto, fue necesario manejar todas las posiciones como traslaciones desde el origen del diagrama.

11.3.4 Orientación a Objetos

Si bien JavaScript es un lenguaje que incluye la definición de objetos, este no incorpora muchas de las características típicamente encontradas en otros lenguajes orientados a objetos. En particular, JavaScript, no tiene un mecanismo formal para manejar herencia entre clases de objetos y mucho menos para crear interfaces o clases abstractas. Sin embargo existen varias librerías y frameworks, entre ellos **JQUERY**, que implementan mecanismos que buscan reproducir algunos de los comportamientos de herencia que incorporan otros lenguajes.

Según el diseño de **MOBIZ** mostrado en el Diagrama 1 se pretende utilizar una clase abstracta y herencia de clases. En este caso se necesitaban sólo las características de reutilización que ofrece el uso de estos mecanismos, y no así otras características típicamente asociadas a herencia como la creación de subtipos o el polimorfismo.

Herencia

JAVASCRIPT permite la definición de objetos mediante la definición de su prototipo. Un prototipo es, en el fondo, simplemente una estructura del tipo diccionario donde se registran las funciones y campos de una clase asociadas a un nombre en particular. De esta forma, para poder implementar la reutilización se puede partir por diseñar mecanismos que permitan copiar las funciones y atributos de un prototipo a otro, y el poder luego sobrescribir algunas de estas funciones en el nuevo prototipo. Para hacer esto, **JQUERY** ofrece el método `jQuery.extend`, que precisamente lo que hace es copiar todo lo de un diccionario a otro, sobrescribiendo en el diccionario de destino cualquier elemento que también este definido en el diccionario de origen. De esta forma, si, por ejemplo, se quisiera crear una relación de herencia en donde la clase *SubClase* herede de la clase *SuperClase* se siguen los siguientes pasos:

```
$.extend(SubClase.prototype,SuperClase.prototype);
$.extend(SubClase.prototype,{elementos nuevos o a redefinir});
```

Otra característica que con la que se deseaba contar era la posibilidad de invocar desde *SubClase* a la versión original de algún método ya sobrescrito. Para esto lo que se hace es usar el método `apply()` que utiliza un enfoque de Continuation Passing Style, permitiendo definir el contexto de ejecución de la función. Siguiendo entonces el caso anterior, un ejemplo de este tipo de invocación sería el siguiente:

```
SubClase.prototype.metodoSobrescrito=function(...){
    ...
    SuperClase.prototype.metodoSobrescrito.apply(this,arguments);
    ...
}
```

Clases abstractas

En la implementación de **MOBIZ** se hizo especial énfasis en la definición de características de interacción y funcionalidades en desmedro de la implementación de algunos de los artefactos del estándar **BPMN 2.0**. Es por esto, que el contar con una clase abstracta que defina la interfaz con la que la aplicación interactúa con estos artefactos es muy relevante. En primer lugar, permite reutilizar gran cantidad de código, pues muchas de las funciones están ya definidas en la clase abstracta. En segundo lugar, deja claro cuales son las partes que hay que definir específicamente para cada artefacto.

En **JAVASCRIPT** no existe el concepto de método abstracto, pero se puede obtener un resultado similar al implementar un método en la clase abstracta que arroje un error que indique explícitamente si la clase que hereda no lo implementa. En concreto, un método “abstracto” se implementaría de la siguiente forma.

```
SubClase.prototype.metodoAbstracto=function(...){
```



```
throw "metodoAbstracto no implementado";  
}
```

Si bien esto no obliga al programador a definir el método en tiempo de compilación, al menos, el error arrojado es claro y es fácil de rastrear su origen usando las consolas de depuración JavaScript tradicionales de los navegadores.

11.3.5 Flechas

Una tarea especialmente desafiante durante la implementación de **MOBIZ** fue la representación de las transiciones como flechas. Por simplicidad se decidió que no se incluirían mecanismos de interacción que permitieran al usuario definir en detalle la trayectoria o forma de la flecha, sino que éstas serían definidas automáticamente. Luego las tareas relevantes que se identificaron para la implementación de esto son tres.

Definición de coordenadas

La definición de las coordenadas de los puntos inicio y término de la flecha se abordó usando puntos de anclaje. Un punto de anclaje se definió como puntos posibles que se pueden usar como los extremos de una flecha que pretende conectar dos artefactos. Cada artefacto, en función de su forma, posición y tamaño, es capaz de entregar cuando se le pide una lista de puntos de anclaje, que consisten en unas coordenadas y un vector normal que indica la dirección contraria al centro del artefacto, tal como se muestra en la Figura 22.

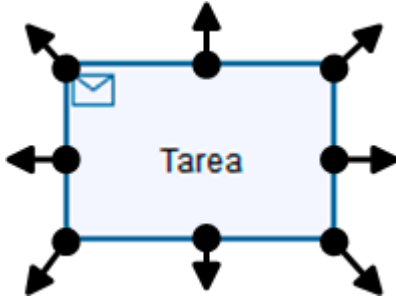


Figura 22. Puntos de anclaje y sus normales.

Luego, si se desea crear una transición entre dos artefactos, cada uno entrega su lista de puntos de anclaje y se decide por uno de cada uno para la flecha. Los criterios para elegir los puntos de anclaje son los siguientes:

1. Solo se puede escoger un par de puntos de anclaje que tengan al menos una de las coordenadas de sus normales con signos diferentes.
2. Un par de puntos de anclaje más cercanos entre ellos tiene mayor prioridad que un par de puntos más distantes.

3. Un par de puntos de anclaje tiene menos prioridad entre más flechas estén asociadas a alguno de sus puntos.

Finalmente, para mantener las flechas fijas a los artefactos, cada uno de estos tiene almacenada una lista de todas las flechas desde o hacia alguno de sus puntos de anclaje. De esta forma, si algún artefacto se mueve o cambia de tamaño, todas las flechas relacionadas son recalculadas y dibujadas acorde.

Dibujo de la flecha en SVG

Una vez que se tienen las coordenadas de inicio y término de la flecha, para dibujarlo se utilizan dos elementos de *SVG*. Lo primero es una línea recta simple que se dibuja desde el punto de origen al punto de término, para lo que se utiliza el elemento para dibujar caminos `<path/>` de *SVG* de la siguiente forma:

```
<path d="M x_inicio y_inicio L x_fin y_fin" />
```

Lo segundo es usar lo que en *SVG* se llaman marcadores. Un marcador es alguna figura que se dibuja al principio, final y/o a lo largo de algún camino. En este caso se definió una figura que es una punta de flecha y que tiene la misma orientación que el camino en la que se coloca de la siguiente forma:

```
<marker id="arrow" viewBox="0 0 10 10" refX="3" refY="5" orient="auto"
markerWidth="10" markerHeight="10">
<path d="M 0 1 L 6 5 L 0 9 L 2 5 z">
</marker>
```

De esta forma, si se coloca este marcador al final de la línea, esta se ve como una flecha apuntando hacia el punto de destino.

Texto de la flecha

La última parte de la implementación de las transiciones fue poder colocar textos asociados a las flechas. Este problema se reduce a simplemente encontrar una buena posición de inicio de la escritura del texto. Se decidió que se quería que el punto de inicio del texto estuviera en una ubicación que estuviera:

- Cerca del punto de inicio de la flecha, pues en **BPMN** el texto de una transición normalmente se refiere a la condición para elegir un camino u otro desde un mismo origen, por lo que tiene sentido que esté cerca de dicho artefacto.
- Debe ser una posición que permita que el texto, en lo posible, no se superponga con la flecha ni con los artefactos de origen o destino.

Para encontrar un punto así se definió una simple heurística que, sin recurrir a detecciones costosas de colisiones, logró definir un punto suficientemente bueno para colocar el texto. Dicha heurística consiste en:

1. Encontrar un vector normalizado $n1$ en la dirección de la transición.
2. Encontrar un vector normal $n2$ perpendicular al vector anterior. Siempre se debe elegir el vector perpendicular que apunte hacia la derecha, salvo en el caso de que el vector normal $n1$ sea horizontal donde es irrelevante cual de los vectores perpendiculares se escoge.
3. Se define un vector directriz d como la suma de los vectores $n1$ y $n2$, multiplicado por una constante de distancia fija. El punto escogido es la suma entre el punto de origen de la transición y el vector directriz d (ver Ilustración 1).
4. Finalmente, se maneja el caso especial donde el punto escogido esté a la izquierda del artefacto, pero no sobre ni debajo de este, de forma que el texto y el artefacto podrían superponerse. En este caso se desplaza el punto encontrado anteriormente en forma paralela a la transición y en la misma dirección, hasta que el nuevo punto este suficientemente a la izquierda como para que quepa el texto, o suficientemente arriba o abajo como para que no pueda colisionar con el artefacto de origen. En el caso en que la distancia entre los artefactos de origen y destino no sea suficiente para esto, el punto solo se desplazara lo más posible (ver Ilustración 2).

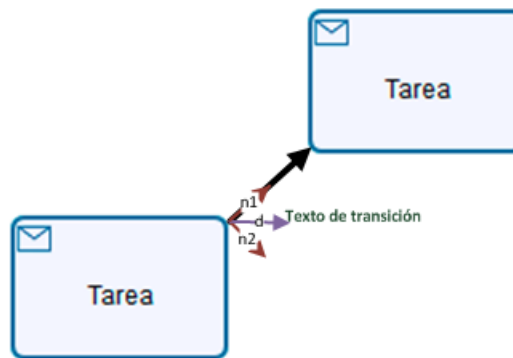


Ilustración 1: Ubicación texto en transición caso trivial



Ilustración 2: Ubicación texto en transición caso no trivial

Capítulo III

Framework de comunicaciones

12 Introducción, alcance y objetivos

La segunda etapa del proyecto consistió en transformar la aplicación monousuario ya construida, descrita en el capítulo anterior, en una aplicación multiusuario, que permita que varios usuarios participen concurrentemente en la elaboración y edición de un mismo modelo.

El primer paso lógico en este respecto fue buscar alguna librería o trabajos ya realizados en los que basarse para realizar la adaptación de la forma más sencilla, eficiente y robusta posible. Al hacer esto se optó por utilizar un paradigma llamado de *Objetos Acoplados*, que estaba específicamente pensado para ser usado en la adaptación de aplicaciones ya existentes para transformarlas en aplicaciones concurrentes. Tanto es así que se estimó que una buena implementación del paradigma podría ahorrar muchísimo tiempo en posteriores adaptaciones y modificaciones para hacer de la aplicación una aplicación concurrente.

Es por esto que se decidió extender un poco el alcance y objetivos de esta etapa para incluir la implementación de un framework de comunicación tan completo y genérico como sea posible en el tiempo disponible, que se pueda proyectar más allá de los límites de esta sola aplicación.

De esta forma, los resultados obtenidos de esta etapa de desarrollo fueron los siguientes:

1. Un framework para el desarrollo de aplicaciones colaborativas utilizando el paradigma de objetos acoplados.
2. Un servidor, implementado con el framework antes mencionado, que da soporte a la comunicación y persistencia necesaria por **MOBIZ**.
3. Una API JavaScript para HTML5 para la implementación de aplicaciones concurrentes y que utilice el servidor antes mencionado para gestionar la persistencia de los modelos y la comunicación entre los participantes.
4. La adaptación de **MOBIZ** utilizando la API mencionada anteriormente, transformándola así en una aplicación colaborativa y concurrente.

En este capítulo se discutirán los principios y ventajas del paradigma de *Objetos Acoplados* y se planteará el diseño de la solución a implementar junto con la estrategia utilizada para definirlo.

13 Objetos Acoplados

El concepto de *objetos acoplados* consiste básicamente en que objetos de la interfaz de la aplicación (AI), en distintos clientes, están acoplados entre si, de forma que si un evento sucede en uno de ellos, este es propagado al resto.

Este enfoque tiene varias ventajas relevantes para la construcción de **MOBIZ**, siendo las principales:

- **Arquitectura replicada:** Pese a utilizar un servidor centralizado para administrar mas fácilmente y en forma efectiva la sincronización, utiliza, además, arquitectura replicada. Esto quiere decir que cada cliente tiene toda la información necesaria para trabajar incluso en ausencia del servidor. Esto, además de la evidente ventaja de poder trabajar offline, ofrece ventajas importantes al intentar sincronizar aplicaciones ya construidas, pues esto se realiza mediante el acoplamiento de sus partes. También es especialmente útil, si la interfaz de la aplicación a sincronizar es considerablemente compleja, como será en el caso de **MOBIZ**, caso en el cual es muy útil poder trabajar en ésta por separado, sin preocuparse de la comunicación.
- **Sincronización dinámica:** Los objetos debieran poder acoplarse y desacoplarse dinámicamente en cualquier punto durante la vida de la aplicación. Esto es especialmente útil para aplicaciones como **MOBIZ**, donde los objetos que serán acoplados serán creados y destruidos dinámicamente.

- **Sincronización Parcial:** Debiera ser posible sincronizar cada elemento de la AI por separado, permitiendo así sincronizar aplicaciones con interfaces completamente distintas. Esto es algo muy importante si se considera que **MOBIZ** debe funcionar en dispositivos de características muy diferentes entre ellos, lo que significa interfaces de aplicación y de usuario diferentes.

14 Implementaciones anteriores

Desde que se propuso el paradigma de los objetos acoplados en 1993, ha habido varias implementaciones, cada una con distintas tecnologías, fortalezas y limitaciones. A continuación se expondrá un resumen de estas implementaciones de modo de poder utilizarlas como referentes de la implementación que se realizará para **MOBIZ**.

14.1 *MatchMaker RMI*

La primera implementación que consideraremos de este paradigma es **MATCHMAKER** (Tewissen, Baloian, Hoppe, & Reimberg, 2000) en su versión basada en **RMI**¹⁰. Esta implementación consideraba un servidor, que se montaba en un contenedor **RMI**, y un framework para implementar las aplicaciones clientes.

Esta implementación funciona con el concepto de sesiones, es decir, un usuario se conecta a una sesión y puede sincronizar objetos con otros clientes conectados a la misma sesión. Para poder hacer esto, el servidor ofrecía interfaces vía **RMI** para las operaciones necesarias para el manejo de sesión desde el cliente, con funciones para unirse o retirarse a una sesión, crear nuevas sesiones, consultar el estado de las sesiones, etc.

El acoplamiento se hace a través de una interfaz publicada por el servidor que permitía acoplar dos objetos y desacoplarlos. La sincronización entre dos objetos era basado en estados, y consistía simplemente en enviar el objeto completo al servidor cada vez que este cambiaba, y el servidor se encargaba de propagar esta información al resto de los clientes.

¹⁰ En realidad MatchMaker RMI es la segunda implementación de este framework, siendo la primera una construida en C/C++. Sin embargo la versión RMI de MatchMaker se considera solo una re-implementación de la primera por lo que con analizar la segunda basta para estudiar sus mecanismos, fortalezas y ventajas.

Además contaba con algunos mecanismos extras, como el acoplamiento por jerarquía, que permitía mantener acoplados varios objetos organizados jerárquicamente con solo acoplar el objeto con mayor jerarquía.

14.2 *MatchMaker SOAP*

Un problema intrínseco de utilizar **RMI** es que no funciona bien si hay firewall, **NAT** u otro tipo de elemento intermedio en la red. Es por esto que se hizo una versión de **MATCHMAKER** que re implementaba la capa de comunicación para que utilizara servicios web **SOAP** en lugar de **RMI** (Baloian, Pino, & Jansen, Implementing the Coupled Objects Paradigm for Synchronizing Distributed Applications Through Firewalls, 2007). Funcionalmente era idéntico a la versión **RMI**, salvo que esta vez era el cliente quien tenía que consultar por cambios en un objeto pues **SOAP**, al estar sobre **HTTP**, solo permite iniciar comunicaciones desde el cliente.

Este cambio solucionó efectivamente el problema que causaba **RMI**, pero a costa de una disminución en los tiempos de respuesta en la comunicación.

14.3 *Objetos Acoplados en HTML5*

Recientemente se desarrolló una implementación para acoplar objetos usando **HTML5**. Esta implementación es experimental, por lo que es bastante minimalista en los mecanismos que provee.

El cliente ofrecía las siguientes funcionalidades:

- **Acoplar y desacoplar un objeto:** Al acoplar un objeto se enviaba el estado inicial del objeto al servidor. El desacoplar un objeto solo se hacia a nivel de cliente para que se dejara de enviar y recibir información de este cliente.
- **Enviar el estado del objeto al servidor:** Este mecanismo estaba diseñado solo para objetos del **DOM** de la página **HTML**. Para hacerlo transformaba el objeto, dependiendo de su tipo, en un objeto **JSON** el cual era enviado al servidor. Este envío debía hacerse manualmente en los métodos de control de eventos de los elementos (**onClick**, **onChange**, etc...) o periódicamente.
- **Solicitar el último estado del objeto al servidor:** Este proceso era realizado periódicamente por el cliente y luego usaba la información entregado por el servidor para actualizar el estado de los elementos del **DOM**. Era necesario hacerlo así pues, al igual que en el caso del **MATCHMAKER** con **SOAP**, al usar **HTTP** toda la comunicación debe ser iniciada por el cliente.

El servidor, desarrollado en **PHP** simplemente almacenaba los mensajes enviados por los clientes y los enviaba a quien los solicitara.

15 Implementación propuesta

La implementación que se propuso busca no sólo solucionar el problema del acoplamiento para **MOBIZ**, sino ser una solución genérica de forma que sea utilizable y adaptable para otras aplicaciones.

Dado que todas las implementaciones anteriormente mencionadas fueron exitosamente utilizadas para implementar diversas aplicaciones, se pensó que una implementación basada en su experiencia debiera ser una buena primera aproximación a la solución.

Las características rescatadas de las implementaciones anteriores fueron las siguientes:

- **Acoplamientos de nodos del DOM:** Cuando un browser procesa una página en **HTML**, cada parte de este es representada por un nodo en el **DOM**. Estos nodos pueden ser accedidos y modificados desde JavaScript permitiendo tener absoluto control sobre la estructura de la página web mediante la manipulación del **DOM**. Dado esto, y siguiendo la estrategia propuesta en la implementación en **HTML5** del acoplamiento de objetos, se cree que acoplar los nodos del **DOM** debiera ser suficiente para poder acoplar las interfaces aplicaciones basadas en **HTML**.
- **Sincronización basada en Estados:** Todas las implementaciones de **MATCHMAKER** usan puramente sincronización por estados por lo que parece ser una estrategia válida a seguir,
- **Sesiones:** Una de las cosas que faltaba en la implementación de objetos acoplados para **HTML5** era el manejo de sesiones u otro mecanismo para aislar usuarios entre ellos. Se propone seguir la estrategia de **MATCHMAKER** donde se crean sesiones y los usuarios se pueden conectar a ellas.
- **Protocolo con WebSocket:** Uno de las limitaciones de las versiones en **HTML5** y la **SOAP** del acoplamiento de objetos, es que se debe emular el recibir un mensaje desde el servidor mediante la consulta periódica de mensajes. Sin embargo, **HTML5** define los **WebSockets**, que al ser canales de comunicación bidireccionales permiten que el servidor pueda enviar mensajes a los clientes sin que estos la soliciten.
- **Fácil de cambiar protocolos:** El cambiar el protocolo de **MATCHMAKER** desde **RMI** a **SOAP** requirió un esfuerzo considerable. Se considera entonces que esta versión permita cambiar con el menor esfuerzo posible el protocolo de comunicación.

Con este conjunto de características en mente, se propusieron algunos diseños de como utilizarlos para implementar la comunicación de **MOBIZ**. Al hacer este ejercicio, si bien se llegaba a soluciones válidas, se pudieron notar algunas carencias que dificultaban la tarea, especialmente si se consideraba que se quería una solución que

fuera fácil de emplear en una aplicación ya construida. Los principales problemas encontrados fueron los siguientes:

- **El manejar solo objetos del DOM no es suficiente:** MOBIZ utiliza SVG para el despliegue del modelo y su interacción con el usuario. El problema es que un solo elemento lógico del BPM, como un evento, es representado por hasta 4 elementos gráficos DOM, por lo que un solo mensaje podría transformarse en hasta 4, cuadruplicando la demanda de la red. En conclusión, sería deseable acoplar objetos JavaScript también.
- **Creación de nuevos objetos a acoplar:** Si solo se usa acoplamiento por estados, resulta complejo administrar la creación de nuevos objetos a acoplar. Esto sucede porque la creación de un objeto nuevo debe ser implementada como un cambio de estado de un objeto ya acoplado para que pueda ser propagada a los otros clientes. De esta forma tenemos la necesidad de utilizar un objeto que contenga al menos una referencia a todos los demás, y manejar la creación o eliminación de objetos como un cambio en su estado. Esto podría tener un impacto negativo en el rendimiento si hablamos de una cantidad considerable de elementos, como es en el caso de MOBIZ.
- **WebSockets no está bien soportado por los browser:** Si bien se sigue considerando que WebSockets es la mejor solución para implementar el protocolo de comunicación, en la actualidad el soporte para WebSockets por parte de los navegadores es deficiente o inexistente. Por esto se requiere implementar la comunicación con otro protocolo, e idealmente poder ir cambiando a WebSockets a medida que mejore su soporte en los distintos navegadores.

Al analizar los problemas anteriores, se decidió incluir las siguientes características adicionales a la implementación para el acoplamiento de objetos:

- **Acoplamiento Selectivo:** Se debe permitir elegir al programador si prefiere acoplar nodos del DOM u objetos lógicos JAVASCRIPT. En el primer caso se seguirá la misma estrategia que la anterior implementación en HTML5 donde se utilizan los callbacks de los eventos para acoplar los nodos. En el caso de hacerlo con objetos JAVASCRIPT se utilizarían los métodos de estos para generar los mensajes de acoplamiento. De esta forma se puede enfrentar el problema de objetos muy complejos en su representación grafica pero más simple en su funcionamiento lógico.
- **Sincronización Mixta:** Debe soportar tanto la sincronización por eventos como por estado. Con esto se soluciona el problema indicado sobre la creación de nuevos objetos, pues esto puede ser modelado como un evento en lugar de un cambio de estado.
- **Múltiples Protocolos:** Asumiendo una adopción progresiva de WebSockets, navegador por navegador, se espera que en algún instante deban convivir ambos protocolos, por lo que esto debe estar considerado.

En la siguiente tabla se resume como las características adicionales propuestas dan solución a los problemas detectados.

| Problema detectado | Solucionado por. |
|---|------------------------|
| El manejar solo objetos del DOM no es suficiente | Sincronización Mixta |
| Creación de nuevos objetos a acoplar | Acoplamiento Selectivo |
| WebSockets no está bien soportado por los browser | Múltiples Protocolos |

16 Diseño de la solución

16.1 API propuesta

Como el foco principal del trabajo no es la construcción de este framework, la definición de la API detallada que se necesita construir a priori no es tan relevante pues esta se puede ajustar por ahora a las necesidades más particulares del proyecto.

Sin embargo es necesario definir las características generales que se proponen debe tener la API con la que se interactuará con el servidor.

Siguiendo las implementaciones anteriores, la sincronización se realizará con una interfaz que permita enviar mensajes entre el cliente al servidor y viceversa. A esta interfaz se le llamó adaptador y tiene la siguiente estructura.

| Modificador y Tipo | Método y Descripción |
|--------------------|--|
| public void | sendToClient(Message message) Método utilizado por el servidor para enviar un mensaje al cliente. |
| public Boolean | sendToServer(Message message) Envía un mensaje al servidor para que sea procesado. |
| public void | couple(String objectId) Indica que el objeto representado por objectId está acoplado a través de este adaptador. |
| public void | decouple(String objectId) Indica que el objeto representado por objectId ya no está acoplado a través de este adaptador. |
| public void | decoupleAll() Desacopla todos los objetos antes acoplados por este adaptador. |

Además, aprovechando las capacidades de JavaScript, se utilizara una API de mayor nivel, construida sobre la anterior interfaz, cuyo objetivo es evitar lo más posible intervenir el código de los objetos con código de sincronización. Esto no sólo es

importante para mantener una separación entre la lógica de negocio y sincronización, sino que también es muy relevante al momento de intentar acoplar aplicaciones ya construidas.

Para hacer esto se usaran dos artefactos, ObjectHandlers y EventHandlers. El primero busca intervenir objetos JavaScript mediante la modificación de sus métodos en tiempo de ejecución, inyectando así la lógica de comunicación. El segundo busca inyectar la lógica de comunicación como un manejador de eventos (por ejemplo el evento onClick de un botón). Más detalles sobre estos artefactos se pueden ver en la sección 16.5.3.

De esta forma, para iniciar la sincronización de objetos se deben seguir los siguientes pasos:

1. Inicializar el adaptador ClientAdapter:

```
var adapter=new ClientAdapter("http://server:port/entry/point");
```

2. Se crea uno o más handler por objeto a acoplar.

- Para objetos JavaScript

```
var handler1=new ObjectHandler(id, target, methods, type);
```

- Para objetos con callbacks

```
var handler2=new BindingHandler(id, target, methods, type);
```

3. Se registran en el adaptador

```
adapter.addHandlers([handler1,handler2]);
```

16.2 Tecnología

Dado que **MOBIZ** esta implementado como una aplicación web, se consideró una buena idea el utilizar el mismo servidor web que se utilizó para la publicación de los contenidos web utilizados. Por esto se escogió utilizar Java con un servidor de aplicaciones **J2EE**, que entrega una plataforma tecnológica sólida tanto para el desarrollo de los componentes web como para el servidor de los objetos acoplados.

16.3 Conceptos generales

Para el manejo de la comunicación se utilizó una arquitectura de cliente/servidor. La base del diseño de la comunicación entre el cliente y el servidor será la misma utilizada por **MATCHMAKER**, en la que el servidor publica dos interfaces para que sean utilizadas por el cliente: la de manejo de sesiones y la de sincronización de objetos. Un esquema general de como se compone una aplicación que utilice este mecanismo es el siguiente:

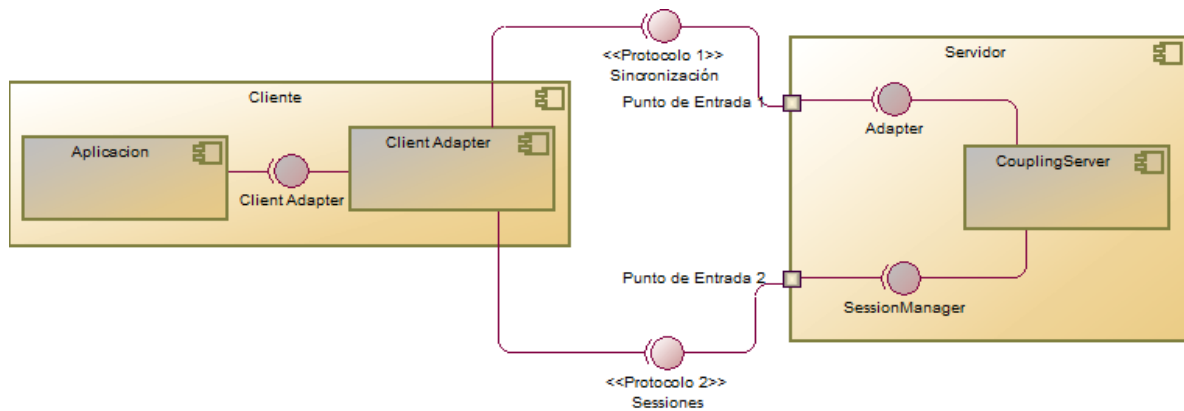


Diagrama 2: Componentes Comunicación

16.4 El servidor

La primera decisión de diseño sobre el servidor fue el construirlo como un framework para la implementación de servidores de objetos acoplados. De esta forma se puede usar como base para la implementación de servidores específicos para cada caso. Las razones para esta decisión son las siguientes:

- Como se mencionó anteriormente, se busca poder crear un servidor que no sólo sirva para **MOBIZ**, sino que de una solución más general al problema del acoplamiento de objetos. Sin embargo este no es el foco de esta memoria y la limitación en el tiempo con que se dispone no permite profundizar lo suficiente para poder definir e implementar un servidor más completo. Sin embargo, al establecer los principios e implementarlos como un framework permite que el trabajo realizado pueda servir como base a trabajos futuros generando implementaciones cada vez más completas.
- Se desea que para un desarrollador sea fácil el integrar varios protocolos, por lo que al menos habría que proveer una pequeño framework para hacerlo.
- Al incluir sincronización basada en eventos, una de las principales dificultades a enfrentar es el mantener un estado único de cada objeto en el servidor, dado que no se puede asumir mucho sobre el efecto de un evento cualquiera sobre un objeto particular. Si bien no se descarta poder implementar un mecanismo genérico y adaptable solo por configuración, como primer acercamiento se prefirió que esta adaptación se hiciera programáticamente mediante la utilización de elementos del framework.
- **MATCHMAKER** proveía mecanismos para la sincronización de casos especiales sólo mediante la adaptación de la aplicación cliente. Sin embargo, al facilitar la adaptación también del servidor, se reduce las necesidades de modificar la aplicación cliente y entrega nuevas alternativas, lo cual se alinea con la idea de un mecanismo que permita transformar una aplicación monousuario en multiusuario fácilmente.

El framework implementado se llama *CouplingServer*. El principal componente del framework es *CouplingManager*, que se encarga de administrar el flujo de los mensajes, procurando que sean validados, persistidos y propagados a los clientes que correspondan. En el Anexo C se expone en detalle la implementación de este y otros componentes del framework.

En la siguiente sección se expone más en detalle las tareas y objetivos de cada componente del framework, y se define la manera en que deben ser utilizados para poder dar forma a un servidor de acoplamiento de objetos completo. Posteriormente se muestra la forma en que esta estrategia se puede utilizar para construir el servidor de **MOBIZ**.

16.4.1 Implementación de un servidor cualquiera

Definición de Protocolos, Puntos de entrada y Adaptadores

El primer paso a seguir es la definición del o los protocolos a utilizar. Una vez definidos, para poder implementar la comunicación en este protocolo, se deben implementar dos componentes que serán las encargadas de recibir e interpretar los mensajes del cliente para que estos puedan ser procesados por el servidor. Estos componentes son los Adaptadores y los Puntos de Entrada.

Un *Adaptador* es una clase java que implementa la interfaz *Adapter* que tiene la siguiente estructura.

| Modificador y Tipo | Método y Descripción |
|--------------------|--|
| public void | sendToClient(Message event) Método utilizado por el servidor para enviar un mensaje al cliente. |
| public boolean | sendToServer(Message event) Envía un mensaje al servidor para que sea procesado. |
| public void | couple(String objectId) Indica que el objeto representado por objectId está acoplado a través de este adaptador. |
| public void | decouple(String objectId) Indica que el objeto representado por objectId ya no está acoplado a través de este adaptador. |
| public void | decoupleAll() Desacopla todos los objetos antes acoplados por este adaptador. |

Un *adaptador* es el componente con el que el servidor recibe y envía los mensajes de y hacia los clientes. La interfaz provista por un adaptador es publicada hacia los clientes mediante un punto de entrada.

Un punto de entrada es una interfaz de comunicaciones, en un protocolo determinado, que publica las funciones necesarias para que el cliente utilice alguna interfaz (en este caso la interfaz *Adapter*) subyacente. Un punto de entrada se diferencia de los demás de cara al cliente por su **URL**. De esta forma, dos puntos de entrada para distintos protocolos pueden coexistir en el mismo servidor y ser accedidos en distintas **URL**. De

esta forma, si bien los adaptadores son la estructura diseñada para poder implementar distintos protocolos, es el concepto de los puntos de entrada el que permite que varios de estos puedan convivir juntos: Distintas **URL** son distintos Puntos de Entrada que podrían usarse para distintos protocolos.

Cabe mencionar que durante el diseño de esta estructura, se considero que era válido tanto una estrategia de publicación directa de cada método de la interfaz que representa, lo que se llamará punto de entrada pasivo (Diagrama 3), como el utilizar el punto de entrada activo como un nivel de abstracción extra que orqueste o modifique las funciones de la interfaz subyacente en nuevas funciones más acordes con el protocolo que implementen, lo que se llamara punto de entrada activo (Diagrama 4). Por ejemplo, dependiendo del protocolo, podría ser necesario incluir el identificador del cliente en cada llamada.

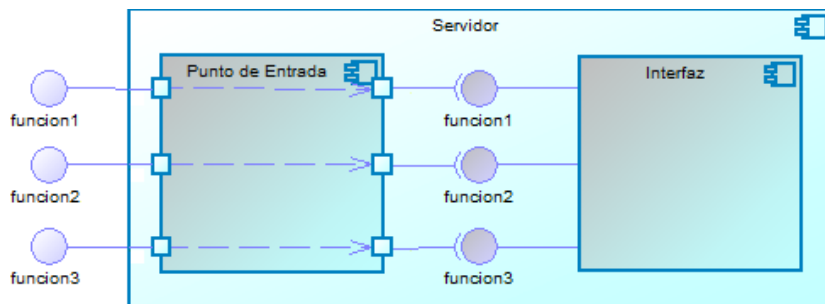


Diagrama 3 : Punto de entrada pasivo

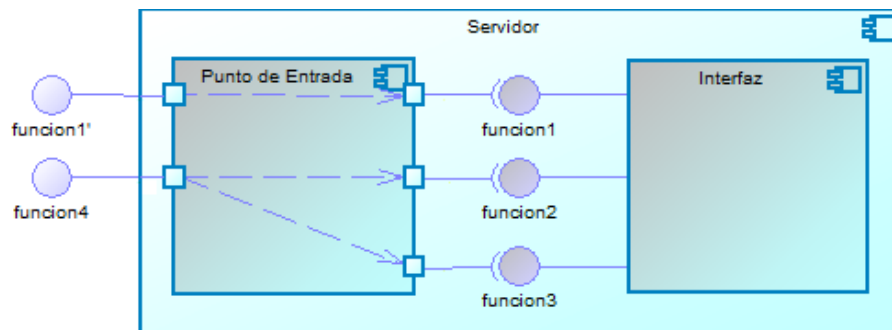


Diagrama 4 : Punto de entrada activo

Definir estrategia de persistencia y validaciones.

Según se mencionó anteriormente, se desea que el servidor soporte acoplamiento tanto basado en estados como basado en eventos, lo cual dificulta la tarea de mantener, y luego persistir, un único estado de cada objeto en el servidor. Para enfrentar esto el framework provee un mecanismo mediante la implementación de un administrador de estado específico para cada caso. La tarea de este administrador es, por un lado, procesar cada mensaje que reciba y utilizarlo para actualizar el estado de los objetos

pertinentes, y por otro lado generar los mensajes necesarios para la actualización del estado de un objeto determinado cuando este es acoplado por cualquier adaptador. Para este propósito, el framework ofrece una clase abstracta llamada `StateManager` que tiene la siguiente estructura:

| Modificador y Tipo | Método y Descripción |
|--------------------------------|--|
| abstract public void | <code>processMessage(Message event)</code> Método utilizado por el servidor para enviar un mensaje al cliente. |
| abstract List<Message> void | <code>getPersistantState (String objectId)</code> Entrega el o los mensajes necesarios para indicar el estado actual del objeto identificado por <code>objectId</code> . |
| abstract public void | <code>persist()</code> Persiste en disco el estado de cada objeto. |
| public boolean | <code>validateMessage(Message message, Adapter adapter)</code> Valida si el mensaje enviado es válido. Un mensaje es valido si cumple con las condiciones definidas por cada validador agregado al <code>StateManager</code> . Solo valida mensajes de eventos. |
| public void | <code>addValidator(Validator validator)</code> Agrega un validador para ser usado en la validación de mensajes. |

Además de la clase abstracta anteriormente mencionada, `CouplingServer` incluye dos implementaciones de `StateManager` listas para usar:

- `BasicEventStateManager`: Esta implementación básica esta pensada para ser muy sencilla de utilizar, pero también es limitada en cuanto a los escenarios en que puede utilizarse. Su mecanismo de persistencia consiste en mantener el último estado enviado para cada objeto y la lista de todos los eventos enviados posteriormente a dicho estado. De esta forma, si un nuevo cliente acopla uno de estos objetos, se le enviará un mensaje con el último estado (de existir) y posteriormente un mensaje por cada evento en el mismo orden que fueron recibidos originalmente.
- `PersistentObjectStateManager`: Este administrador es mucho más versátil que el anterior, pero también es más complejo de configurar. Este administrador mantiene una lista de todos los objetos acoplados, cada uno de estos representado por un objeto que se encargará de ejecutar los eventos y cambios de estado que le correspondan. Para esto, cada objeto debe implementar los métodos para ejecutar cada evento relevante que se le envíe, además de implementar la interfaz `PersistentCoupledObject` que tiene la siguiente estructura:

| Modificador y Tipo | Método y Descripción |
|----------------------|---------------------------------|
| public List<Message> | <code>getStateMessages()</code> |

| | |
|---------------|--|
| | Entrega la lista de mensajes que definen el estado del objeto. |
| public String | getModelId() Entrega el ID con el que se diferenciara al objeto entre todos los otros objetos que implementen esta interfaz. |
| public void | updateState(Map<String, Object> state) Actualiza el estado del objeto. |

Definición de estrategia de manejo de sesiones, su protocolo y punto de entrada

Como mecanismo para agrupar distintos clientes entre sí y aislar estos grupos entre ellos, se utilizó el mismo concepto que utilizaban **MATCHMAKER RMI** y **SOAP** que llamaron sesiones. De esta forma, un cliente puede crear una nueva sesión o unirse a una sesión ya creada, y los mensajes solo serán propagados entre clientes que pertenezcan a la misma sesión. Para esto, cada sesión cuenta con un **CouplingManager** independiente, y por ende su propio **StateManager** y conjunto de validadores, por lo que dos sesiones distintas pueden tener un comportamiento totalmente distinto entre ellas. Por esta razón el administrador de sesiones se presenta como una Interfaz que debe ser implementada según cada caso. Dicha interfaz se llama **SessionManager** y tiene la siguiente estructura:

| Modificador y Tipo | Método y Descripción |
|--------------------|--|
| public Client | createClient() Crea un nuevo cliente. |
| public Client | getClientById(Long clientId) Obtiene el cliente asociado a clientId. |
| public Session | createSession(Object...options) Crea una nueva sesión. La clase que implemente esta interfaz debe decir cuales son las opciones que utilizará. |
| public Session | getSessionById(Long sessionId) Obtiene la sesión asociada a sessionId. |
| public Session | getClientSession(Client client) Obtiene la sesión a la que está unido el cliente indicado. |
| public void | destroyClient(Client client) Elimina el cliente indicado. |
| public void | destroySession(Session session) Elimina la sesión indicada. |
| public void | joinClientToSession(Client client, Session session) Une el cliente a la sesión indicada. |
| public void | removeClientFromSession(Client client) Retira el cliente de la sesión a la que pertenece. |

Siguiendo la misma estrategia que para los adaptadores, esta interfaz es publicada hacia el cliente por medio de un punto de entrada, ya sea activo o pasivo, a través del protocolo que se prefiera.

16.4.2 Implementación servidor Mobiz

Protocolo

La especificación actual de **HTML5** incluye los llamados *WebSockets*. Estos son sockets que pueden abrirse desde el cliente, que permite la comunicación bidireccional entre el cliente y el servidor. Sin embargo, en la mayoría de los navegadores actuales, la implementación de *WebSocket* es limitada o inexistente. A falta de algún mecanismo bidireccional de comunicación, la siguiente elección más lógica es utilizar un protocolo basado en **HTTP**, pues tanto los browsers como el servidor web a utilizar lo soportan en forma nativa y robusta. Entonces, se utilizarán solicitudes **HTTP** de tipo **POST** y **GET** para enviar y consultar información al servidor.

Adaptador

El adaptador construido para este protocolo se llamará *JSONBufferAdapter*. Como su nombre lo sugiere, las dos principales características son:

- Puede recibir los mensajes en formato **JSON** en lugar de un objeto tipo *Message*, y realiza internamente la transformación.
- Como no puede enviar directamente la información al cliente, el método *sendToClient* escribirá el mensaje en un *Buffer* también en formato **JSON**. Luego, a través de un método público, el punto de entrada puede consultar y extraer el contenido de este buffer a solicitud del cliente.

Punto de entrada

El punto de entrada para los adaptadores será un Servlet llamado *CouplingServlet*. Este Servlet estará mapeado a tres **URI** distintas:

- **/coupling*: Esta **URI** se utiliza para enviar y consultar mensajes al servidor. Si la solicitud es del tipo **GET**, la respuesta del servidor será una lista de mensajes con los eventos y cambios de estados que se deben enviar al cliente. Si es del tipo **POST**, se espera que en el atributo de la solicitud *message* contenga un mensaje en formato **JSON** que será enviado al servidor. En este último caso, si el mensaje enviado por el cliente es un evento, la respuesta del servidor puede tener un código de respuesta **HTTP** distinto a 200 para impedir que el evento se ejecute en el cliente. El formato en que se envía la respuesta es el definido por la especificación de **HTML5** para ser usada por una **API** que ofrece para emular eventos enviados por el servidor mediante llamadas **AJAX** periódicas.
- **/coupling/couple*: Esta **URI** se utiliza para acoplar un objeto. Se *espera* una solicitud del tipo **POST** que contenga el identificador del objeto en el atributo *objectId* de la solicitud. Solo el código **HTTP** de la respuesta es relevante para el cliente, con 200 indicando éxito y cualquier otro código indicando error.

- `*/coupling/decouple`: Esta **URI** se utiliza de la misma forma que la anterior, con la diferencia que el objeto indicado por el atributo `objectId` será desacoplado para el cliente que solicita. Solo el código **HTTP** de la respuesta es relevante para el cliente, el resto es descartado, con 200 indicando éxito y cualquier otro código indicando error.

Persistencia

La estructura de clases construida para administrar la persistencia es la siguiente:

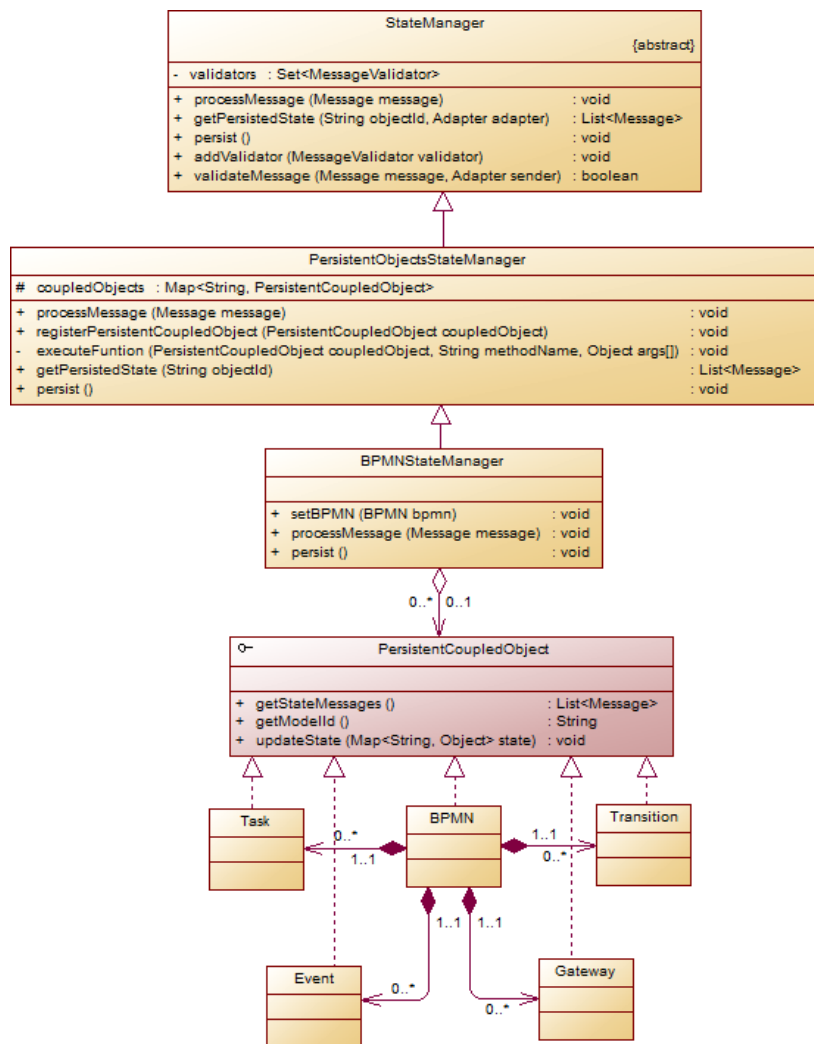


Diagrama 5: Clases persistencia BPMN Server

Cada uno de los objetos que heredan de `PersistentCoupledObject` tiene la misma interfaz que el objeto del cliente que representan, de forma que los mismos eventos pueden ser ejecutados en cada uno de ellos. Usando una implementación de **JPA** llamada `ObjectDB`,

cada uno de estos objetos es persistido automáticamente en lugar de usar la persistencia bajo solicitud mediante el método `persist`. Además, el administrador de estado, que hereda su comportamiento de `PersistentObjectStateManager`, puede cargar todos los objetos persistidos para un BPMN automáticamente y sólo permite un BPMN a la vez por instancia.

Sesiones y clientes

En las aplicaciones web, cuando un usuario se conecta, se le crea una sesión propia, en donde se puede registrar toda la información sobre el usuario que se necesite durante la interacción con la aplicación. Esta sesión se conoce como *Sesión HTTP*. El manejo del ciclo de vida de las sesiones HTTP es realizado por el servidor, en el caso de J2EE el *Servlet Container*, y consiste en mantenerla viva mientras el usuario siga interactuando con la aplicación, y eliminarla si pasa mucho tiempo sin hacerlo. El servidor puede saber que usuario realiza cada solicitud y por ende, cual es la sesión HTTP que le corresponde.

En cambio, *CouplingServer* maneja el concepto más genérico de cliente en lugar de usuarios, y las sesiones son compartidas entre varios clientes en lugar de estar asociados a un solo usuario.

Luego, para poder aprovechar las características del servidor web y solucionar los conflictos con los nombres de los objetos, se utilizó la siguiente estrategia:

- Se creó la clase `User` que extiende de la clase `Client`, agregando algunos campos adicionales sobre el usuario, como el nombre del usuario.
- Se creó la clase `WorkingBoard` que extiende de la clase `Session` y agrega como un campo adicional el BPMN sobre el que los usuarios conectados a esta sesión están trabajando.
- Se creó un `SessionListener`, llamado `UsersListener`, que se encarga de que cada vez que una sesión HTTP es creada se cree un usuario asociada a esta, y de que si la sesión HTTP es destruida, el usuario correspondiente también lo sea.
- Se creó la clase `AccessManager` que implementa la interfaz `SessionManager` que siguiendo el patrón *singleton* y que contiene las referencias a todos los clientes y sesiones. Esta implementación se encarga de eliminar las sesiones para que no hayan usuarios conectados y poder crearlas automáticamente desde disco si un usuario quiere conectarse a ésta. Esto, junto con el uso de `UsersListener`, permite ligar el ciclo de vida de las sesiones del servidor web, con el de las sesiones de acoplamiento de objetos, dejando así que sean administradas por el servidor de aplicaciones.

Punto de entrada para administración de sesiones

Siguiendo el paradigma de las aplicaciones web, un usuario puede solo conectarse al cliente en una nueva sesión o conectarse a una sesión ya creada, y la administración de la creación, eliminación y asignación de sesiones a los usuarios es manejado por el servidor y es transparente para el cliente.

Para hacer esto, se utilizó un servlet con la URI `*/bpmn`. Este servlet, llamado `AccessServlet`, al recibir una solicitud del tipo `GET`, revisa si se indicó el identificador del modelo `BPMN` al que se quiere acceder, mediante el parámetro `bpmn` y busca el `WorkingBoard` que le corresponda. Si no se indica un modelo, se crea uno nuevo junto con su `WorkingBoard`. Luego se registra al usuario, obtenido desde la sesión `HTTP`, en el `WorkingBoard` correspondiente y se le responde al cliente con el código `HTML` que implementa el cliente para la aplicación, en donde se incluyen los datos relevantes del usuario y del modelo `BPMN` asignados.

16.5 El cliente

En la siguiente figura se muestra conceptualmente la estructura de clases que componen la capa de comunicación del cliente¹¹:

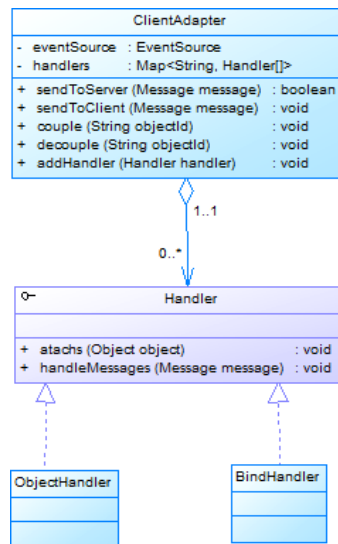


Diagrama 6: Clases Comunicación Cliente

¹¹ El diagrama de clases que se muestran es sólo referencial. JavaScript no tiene la misma estructura de objetos de otros lenguajes, no soporta herencia y es débilmente tipado, por lo que algunas partes de la implementación que usan alguna de estas características no puede ser representadas en un diagrama de clases tradicionales.

16.5.1 ClientAdapter

El principal componente del cliente, tal como se muestra en Diagrama 2 es `ClientAdapter`. Este objeto cumple el mismo propósito que el los adaptadores del servidor, de hecho utilizan la misma interfaz y hay sólo uno por cliente. Los métodos `couple`, `decouple` y `sendToServer` están implementados usando una llamada `AJAX` a las direcciones `%/coupling/couple`, `%/coupling/decouple` y `%/coupling` respectivamente, donde `%` es la `URL` del contexto de la aplicación web (ver página 53).

Para la lectura, se utiliza la `API` de `EventSource`, que es parte de la especificación de `HTML5` y que tiene por objeto el emular eventos enviados por el servidor mediante consultas `AJAX` periódicas. Usando esta `API` la implementación consiste en iniciar un `EventSource`, indicándole que las consultas debe hacerlas a la dirección `%/coupling`, y que el resultado lo debe procesar la función `sendToClient` del objeto `ClientAdapter`.

16.5.2 Validación de mensajes¹²

Es posible que el servidor valide un mensaje enviado por un cliente. Si el servidor decide que el mensaje no es válido se lo informará al cliente. De esta forma, para poder honrar la validación del servidor, en el caso de mensajes de eventos, el cliente envía el mensajes antes de ejecutar el evento localmente, de modo de ejecutarlo sólo si la respuesta del servidor es positiva. Sin embargo, para el caso de los mensajes de estado esto no puede hacerse, pues debe ejecutarse cualquier evento antes de poder generar el nuevo estado. Es por esto que la validación de mensajes de estado no esta actualmente soportada ni en el cliente ni en el servidor.

16.5.3 Handler

La segunda parte de la implementación del cliente es hacer que se generen los mensajes y manejar los mensajes recibido de modo que se complete la propagación de eventos y cambios de estado.

Para esto se utilizaron los llamados manejadores o `Handler`. Cada manejador se adjunta a un objeto `JAVASCRIPT` o elemento del `DOM` (o eventualmente a otro tipo de estructura que se pueda definir) que se quiera acoplar. Los manejadores cumplen un doble

¹² Si bien para este caso no se ha implementado ningún validador, el mecanismo de validación de mensajes se utiliza también para asegurar el orden de propagación de los eventos, por lo que igualmente debe manejarse. Más detalles en el Anexo C.

propósito. Por un lado, se adjuntan al objeto que manejan, interviniendo su funcionamiento de modo que se envíen los mensajes de eventos o cambios de estado. Por otro lado, se registran en `ClientAdapter` y se encargan de realizar las operaciones necesarias en el objeto que manejan cuando se recibe un mensaje que lo requiera.

Un mismo objeto puede tener varios manejadores distintos para poder tener más control sobre como se realiza el acoplamiento.

A continuación se explican más en detalle los dos tipos de manejadores implementados.

ObjectHandler

Este manejador esta diseñado para acoplar objetos **JAVASCRIPT** mediante el acceso a sus métodos. Cuando se adjunta a un objeto es capaz de intervenir las funciones de modo de que se envíe el mensaje correspondiente. Para esto hace uso de la capacidad de JavaScript de crear y asignar funciones en tiempo de ejecución como si de cualquier otro tipo de variable se tratase. A continuación se muestra el pseudo-código de como esto es realizado¹³.

```
//Registra la función original
this.función_original=objeto.funcion;
//Reemplaza la función por una nueva
objeto.funcion=function(argumentos){
  if(clientAdapter.enviarMensaje(this, objeto)){
    //Si el envío es exitoso ejecuta la función
    //original
    return función_original.apply(argumentos);
  }
}
```

Opciones

- `explicitMapping`: Lista con los nombres de las funciones a manejar.
- `mappingExeptions`: Lista de expresiones regulares. Si no se define ninguna función en la opción `explicitMapping` se manejaran todas las funciones salvo las que cumplan con alguna de las expresiones regulares definidas en esta opción.
- `messageType`: El valor de esta opción puede ser `EVENT` (valor por defecto) o `STATE`. Indica si se enviarán mensajes de evento o de estado cuando se ejecute la función.

¹³ No se muestra el código original pues contiene mucha complejidad extra producto del lenguaje y de otras tareas que realiza, como validaciones o evitar que un método para el que ya se envió un mensaje haga una llamada a otro método que genere otro mensaje.

- `getElementState`: El valor de esta opción debe ser una función que entregue un diccionario con los valores relevantes que definan el estado del objeto. Si no se define, la función por defecto trata de ejecutar la función `getState` del objeto y si no esta definida, aprovechando que todo objeto en JavaScript puede ser representado como un diccionario, retorna el objeto como resultado.
- `setElementState`: El valor de esta opción debe ser una función que reciba como entrada un diccionario con los valores que representen un estado del objeto y lo utilice para actualizar el estado del objeto. El comportamiento de esta función por defecto es el utilizar la función `setState` en el objeto para actualizarlo, y si esta no esta definida reemplaza (o agrega) todos los valores del diccionario de entrada directamente en el objeto.

BindHandler

Este manejador tiene por objetivo ocuparse del acoplamiento mediante el manejo de funciones de `callbacks` o, usando la terminología de **JQUERY**, mediante el `binding` de funciones para que se ejecuten cuando sucede un evento determinado. Gracias al uso de **JQUERY** es posible hacer esto indistintamente para nodos del **DOM**, que tiene sus eventos definidos como el `onClick`, o `onChange`, u objetos creados usando **JQUERY** que pueden definir sus propios eventos. La principal diferencia con el otro manejador es que, en el caso de recibir un evento no se puede reproducir el evento mismo, sino que solo pueden ejecutarse las funciones de `callback` registradas al evento con los mismos parámetros.

El mecanismo para adjuntarlo a un objeto es mucho mas simple, pues solo registra una nueva función al principio de la cadena de funciones registradas como `callbacks` del evento que se encarga de la generación de los mensajes.

Opciones

- `bindings`: Lista con los nombres de los eventos en donde se registrará. En el caso de los eventos de los nodos del **DOM** se utiliza la nomenclatura usada por **JQUERY**, donde por ejemplo el evento `onClick` se le llama solo `click`, al evento `onChange` solo `change`, etc.
- `messageType`: El valor de esta opción puede ser `EVENT` (valor por defecto) o `STATE`. Indica si se enviarán mensajes de evento o de estado cuando se ejecute la función.
- `getElementState`: El valor de esta opción debe ser una función que entregue un diccionario con los valores relevantes que definan el estado del objeto. Si no se define, la función por defecto trata de ejecutar la función `getState` del objeto y si no esta definida, aprovechando que todo objeto en JavaScript puede ser representado como un diccionario, retorna el objeto como resultado.
- `setElementState`: El valor de esta opción debe ser una función que reciba como entrada un diccionario con los valores que representen un estado del objeto y lo

utilice para actualizar el estado del objeto. El comportamiento de esta función por defecto es el utilizar la función `setState` en el objeto para actualizarlo, y si esta no esta definida reemplaza (o agrega) todos los valores del diccionario de entrada directamente en el objeto.

Plugin de jQuery: coupler

Con el objetivo de simplificar el uso de la API del cliente, se creo un plugin de **JQUERY** llamado ***coupler***. Este plugin ofrece una interfaz unificada para acoplar un objeto, encapsulando la creación y registro de los handlers, y estableciendo una serie de valores por defecto para las distintas opciones de los handlers.

La función definida por este plugin recibe un único argumento, que es un diccionario con las opciones con las que se quieren crear los handlers para el objeto con la siguiente estructura.

```
{EVENT:{ //Opciones para la configuración de handlers de eventos
bindings:[] //Si esta definido se crea un BindHandler con estas opciones
},
explicitMapping:[] //Si esta definido se crea un ObjectHandler con estas opciones
STATE:{//Opciones para la configuración de handlers de eventos
bindings:[] //Si esta definido se crea un BindHandler con estas opciones
},
explicitMapping:[] //Si esta definido se crea un ObjectHandler con estas opciones
}}
```

17 Transformación de Mobiz en una aplicación concurrente

La estrategia para la transformación de **MOBIZ** en una aplicación concurrente se basa en el acoplamiento de las clases de la capa de lógica de control (más información en el Capítulo II, sección 11.1.2) tal como se ilustra en la Figura 23.

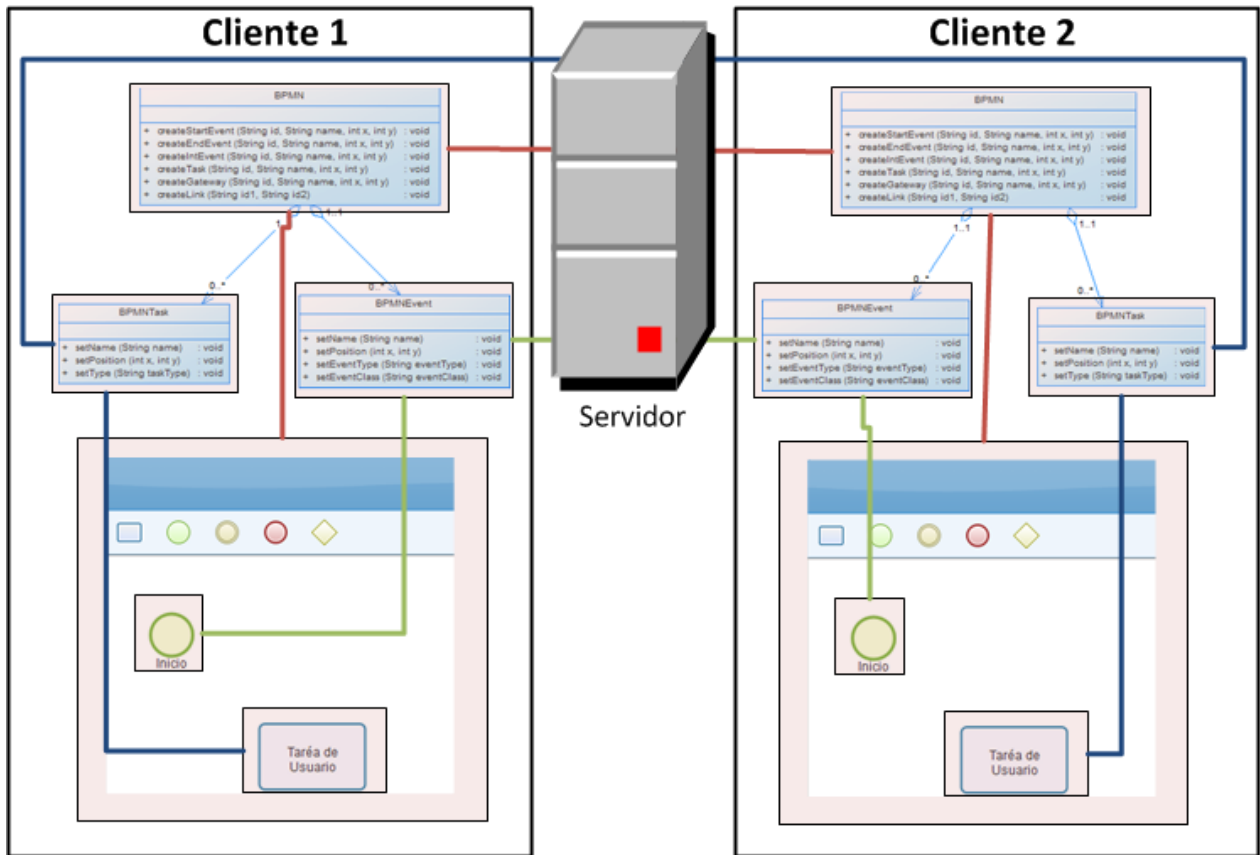


Figura 23: Acoplamiento Mobiz

Tal como se propuesto en un principio, el acoplamiento debe ser selectivo y puede ser mixto, por lo que se debe definir específicamente por cada objeto la estrategia con la que se va a acoplar. Para el caso de **MOBIZ**, la siguiente tabla resume la estrategia utilizada para cada clase, indicando para cada una el tipo de acoplamiento a utilizar y los métodos o eventos de callback que se utilizan para acoplarlos.

| Clase | Tipo de Acoplamiento | Métodos/Callbacks |
|-----------------|----------------------|--|
| BPMN | Métodos/Eventos | createStartEvent, createEndEvent, createIntEvent, createTask, createGateway, createTransition, deleteEvent, deleteTask, deleteGateway, deleteTransition. |
| BPMNTask | Callbacks/Estado | change |

| | | |
|-----------------------|------------------|--------|
| BPMNEvent | Callbacks/Estado | change |
| BPMNGateway | Callbacks/Estado | change |
| BPMNTransition | Callbacks/Estado | change |

17.1 Modificaciones necesarias

Si bien la API para manejar el acoplamiento de objetos esta diseñado para que se necesiten la menor cantidad de cambios posibles en el código de la aplicación original al acoplar los objetos, igualmente puede ser necesario hacer algunas adaptaciones o, en otras variaciones puede ser más simple hacer algunas modificaciones que sobrecargar la configuración de los handlers.

En el caso de **MOBIZ** se hicieron algunas modificaciones en las clases de la capa de lógica de control para simplificar el acoplamiento de éstas.

En primer lugar, en la clase *BPMN*, los métodos de creación de artefactos originalmente creaban automáticamente un identificador único correlativo. Para poder acoplar estos objetos era necesario que se creara un identificador único pero que no tuviera colisiones con identificadores creados por otros clientes. También, para poder acoplar como un evento la creación de los artefactos era necesario que estos métodos recibieran como argumento el identificador y no que se creara dentro del mismo método. Para solucionar estos problemas se hicieron las siguientes modificaciones¹⁴.

- Se creo en la clase *BPMN* un método `_GENERATEID` que, además de un correlativo utiliza el identificador del usuario y de la sesión para generar los identificadores, evitando así que se repitan entre clientes.
- Se modificaron los métodos de creación de artefactos para que recibieran el identificador como entrada, levantando un error si éste no se proveía.
- Se modificaron las invocaciones a estos métodos para que se invocara primero al método `_generateld`.

La segunda modificación que se hizo, si bien no era obligatoria simplifica bastante la configuración del acoplamiento de objetos. Lo que se hizo fue crear en la clase *BPMNElement* los métodos `setState` y `getState`, que como se mencionó anteriormente son los métodos por defecto que se utilizan en los handlers para la generación y actualización del estado. De esta forma, como todos los otros elementos heredan de

¹⁴ Notar que ninguna de las modificaciones mencionadas genera dependencia con el acoplamiento de objetos, por lo que la aplicación sigue funcionando exactamente igual luego de implementarlas.

esta clase y el estado se genera igual para todos los elementos, no es necesario definir esta función para ningún handler.

Una vez realizadas estas modificaciones, y aprovechando la estructura de los plugins de **JQUERY** con la que esta construida **MOBIZ**, el código que transforma la aplicación completa de monousuario a multi-usuario concurrente es el siguiente.

```
//Inicializacion del plugin BPMN. Recibe como parámetro el id de //usuario para la
creación de identificadores
$("#board").bpmn({ indexSuffix : '${user.id}'})
//Acoplamiento del objeto BPMN. Por defecto un ObjectHandler y //acoplando todas los
métodos cuyo nombre no empiecen con _
.coupler()
//Manejo de los eventos que se lanzan al crear algún artefacto nuevo.
.bind('taskCreation eventCreation gatewayCreation transitionCreation',
function(event, element) {
    //Acoplamiento de el nuevo elemento creado por estado.
    $(element).coupler({
        STATE : {
            bindings : [ 'change' ]
        }
    });
});
$.adapter.start();
```

17.2 Acceso a un modelo en particular

Un punto que no se ha mencionado aún es como puede un usuario acceder de nuevo a un modelo en que ya ha trabajado antes, o como se puede unir un nuevo usuario a trabajar en el mismo modelo en que otros usuarios ya estén trabajando. En la versión actual del software esto se puede hacer solamente mediante la administración de las URL de cada modelo.

Cuando uno accede a la aplicación, la URL que se despliega en el navegador incluye el identificador único del modelo en el que se esta trabajando. De esta forma, si se vuelve a acceder a la misma URL, se vuelve a desplegar el mismo diagrama, y si dos usuarios acceden a la misma URL en distintos dispositivos, ambos estarán trabajando en el mismo modelo en forma concurrente.

Capítulo IV

Conclusión y trabajos futuros

18 Análisis de Resultados

En esta sección se intentará exponer, reflexionar y analizar los resultados más relevantes obtenidos durante el trabajo realizado.

Es importante desatacar que, dado el tiempo disponible para realizar el trabajo, la validación exhaustiva de lo desarrollado fue dejada fuera del alcance de esta memoria. Por esta razón, a continuación se expondrá un análisis que no está enfocado a evaluar cuantitativamente que tan buena o mala son las soluciones construidas, sino a evaluar cualitativamente si son suficientemente buenas para los problemas que buscan enfrentar.

18.1 Funcionalidades e interfaz

El principal objetivo de **MOBIZ** es el de apoyar el levantamiento de procesos, de la misma forma en que lo hace **NETSKETCHER**. La validación de si la forma en que **NETSKETCHER** funcionaba era o no una buena solución fue realizada por sus autores de este por lo que se asume que **MOBIZ** también lo es.

De la misma forma la interfaz esta fuertemente basada en la de **BIZAGI PROCESS MODELER**, editor de diagramas **BPMN** muy difundido y utilizado, y en este aspecto se puede ver que **MOBIZ** tiene una interfaz muy bien lograda. **MOBIZ**, tanto en su versión móvil como de escritorio, se siente y responde de forma casi idéntica a como lo hace **BIZAGI PROCESS MODELER**, por lo que es de esperarse que la interfaz no tenga mayores problemas que entorpezcan la usabilidad del producto.

En definitiva, si bien un análisis más exhaustivo podría dejar ver varios puntos que podrían mejorar, se considera que **MOBIZ** es capaz de enfrentar exitosamente la problemática para la que fue diseñado.

18.2 Framework de comunicación

Si bien la construcción del framework de comunicación fue parte importante del desarrollo del trabajo, el análisis detallado de la mayoría de sus características está fuera del alcance de la memoria por no estar relacionado con el objetivo de ésta.

Sin embargo, es importante referirse al menos a los temas de los tiempos de respuesta y manejo concurrencia pues afectan directamente la calidad de **MOBIZ**.

Si bien por razones de tiempo se dejaron fuera análisis cuantitativos sobre dichos puntos, existen varias consideraciones que se tuvieron en cuenta que hacen pensar que el framework resuelve al menos, las necesidades particulares de **MOBIZ**:

- La mayor parte del manejo de concurrencia la hace el servidor de aplicaciones JEE, por lo que la solución es fácilmente escalable aumentando la cantidad o calidad del hardware.
- El caso en estudio de **MOBIZ** considera grupos pequeños de personas (<10) y la interacción concurrente no debiera ser muy intensa, pues el desarrollo de un diagrama de proceso BPMN, especialmente en terreno, no es una tarea rápida.
- Cada artefacto del modelo esta acoplado independientemente y se usan, cada vez que se puede, mensajes de evento en lugar de estado, por lo que la cantidad y tamaño de los mensajes enviados entre clientes está bastante acotado.
- Desde el punto de vista de un solo cliente, los tiempos de respuesta de la aplicación no se ven notablemente afectados al cambiar la aplicación desde su modo monousuario a multiusuario. Esto es gracias a que casi todo el trabajo que realiza el servidor lo realiza asíncronamente y no deja a los clientes esperando respuesta. Por lo mismo es de esperarse que estos tiempos aumenten considerablemente con la cantidad de usuarios concurrentes¹⁵.

¹⁵ Más detalles de como esto es realizado se encuentran en el Anexo C

- Los tiempos de respuesta están relacionados casi exclusivamente con la limitación técnica de poder iniciar comunicaciones desde el servidor y no solo desde los clientes. Actualmente los clientes deben periódicamente realizar consultas al servidor¹⁶, y los tiempos de respuesta que se han observados están acotados entre T y 2T segundos.

19 Conclusión

En esta sección se intentará hacer una última recopilación y resumen de los objetivos iniciales de este trabajo y los resultados obtenidos en hacerlo. Desde ahí, mediante el análisis de estos resultados y del proceso realizado para obtenerlos se expondrán, a modo de conclusión, las principales características que se destacan del proyecto completo, buscando así dejar claro lo que se pudo obtener de éste.

Cuando se planteo el proyecto, el objetivo estaba claro: Se quería contar con una herramienta de modelado BPMN que permita trabajar concurrentemente y en terreno al levantamiento de procesos.

En el momento en que se inicio este proyecto y actualmente, las herramientas de modelado BPMN no son pocas y muchas son de alta calidad. Algunas incluso ofrecen interfaces web y algunas herramientas colaborativas. Sin embargo, la única aplicación que buscaba hacerse cargo de la necesidad planteada era aquella propuesta por Baloian y colaboradores (Baloian, y otros, 2011) llamada **NETSKETCHER**, y cuyos autores logran concluir que su uso efectivamente resolvía la necesidad que existía y aportaba en el levantamiento de procesos.

De esta forma el proyecto a realizar se podía resumir en tomar la solución de **NETSKETCHER** y llevarla un paso más allá, mediante su implementación utilizando los últimos estándares y tecnologías y con interfaces a la par de las aplicaciones modernas de modelamiento. De esta forma se pretendía hacer una nueva aplicación más accesible, completa y usable, transformándola en una herramienta aún más útil para apoyar el levantamiento de procesos.

En definitiva lo que se quería lograr era:

- Un modelador BPMN2.0 que pueda ser accedido vía web por dispositivos móviles, de modo de poder ser utilizado en terreno.

¹⁶ Por limitaciones en el navegador Chrome, este periodo no puede ser menor a 1 segundo. De lo contrario Chrome realiza Bussy Waiting pues cualquier número menor a 1 segundo es asumido como cero por Chrome.

- Concurrente y colaborativo, de forma de permitir la participación simultánea o asíncrona de diversos participantes, tanto analistas expertos en modelamiento, como personas con conocimiento de los procesos a levantar.
- Poder alternar de alguna forma y según la necesidad entre equipos móviles y de escritorio para el trabajo en los modelos BPMN construidos.

Para lograr estos objetivos, las principales decisiones tomadas y que dieron forma al proyecto fueron.

- Basar la interfaz de usuario en la utilizada por **BIZAGI PROCESS MODELER**.
- Utilizar HTML5, jQuery y jQuery Mobile para el desarrollo del cliente.
- Utilizar JEE sobre Tomcat para el desarrollo de los componentes del servidor.
- Usar Objetos Acoplados como la base conceptual y funcional para la implementación de la capa de comunicación.
- Construir la capa de comunicación como una solución mas genérica en lugar de algo mas AD-HOC a Mobiz.

Ahora, al final del proyecto, los resultados obtenidos son:

- Una aplicación para equipos de escritorio, con una interfaz web y concurrente, que permite la creación y modificación de modelos BPMN2.0 (aunque aún faltan algunos artefactos). La interfaz que ésta ofrece tiene poco o nada que envidiarle a las interfaces ofrecidas por otras aplicaciones de modelamiento modernas.
- Mediante la adaptación de la anterior se hizo una interfaz para la misma aplicación pero para equipos móviles, también concurrente. La interfaz esta diseñada para poder ser usada incluso desde *smartphones* y no sólo desde dispositivos más grandes como *tablets*. Además, no sólo ofrece las funcionalidades básicas, sino que todas las funcionalidades disponibles en la aplicación de escritorio están también en esta versión.
- Los modelos se almacenan en el servidor y pueden ser accedidos en cualquier momento e indistintamente desde un dispositivo móvil o de escritorio.
- Como subproducto de la construcción de la capa de comunicación se construyó un framework bastante completo para el manejo de las características de concurrencia y distribución de la aplicación, que podría ser utilizado para otro tipo de aplicaciones.

A continuación se exponen las conclusiones obtenidas a la luz de ver en retrospectiva las decisiones tomadas, del análisis de los productos obtenidos.

La primera conclusión que se pudo obtener del trabajo, es que queda en evidencia la posibilidad de implementar herramientas que normalmente son encontradas en equipos

de escritorio como aplicaciones web. Esto se puede hacer de forma profesional y ordenada, sin que esto signifique mayor dificultad ni comprometer la calidad del producto, gracias a la potencia y madurez de HTML5 y los numerosos frameworks y librerías que hoy existen para hacerlo.

Lo segundo que se pudo observar es que es que HTML5 se perfila potentemente como una alternativa válida no solo en el desarrollo de aplicaciones web con interfaces de alta complejidad, sino como base del desarrollo de aplicaciones multiplataforma. De hecho en este caso, al seguir la estrategia de implementar primero la interfaz en el dispositivo donde es más fácil o natural de usar y luego adaptarlo a otros, se pudo lograr que la segunda interfaz tuviera todas las funcionalidades de la principal y con muy poco esfuerzo. De haber sido construidas en forma independiente lo más probable es que al interfaz móvil hubiese sido más limitada. En definitiva, HTML5 no sólo facilita el desarrollo de aplicaciones para distintas plataformas similares, sino que fomenta y facilita la adaptación de éstas para plataformas con características diferentes, lo cual parece haber sido una gran ventaja en el desarrollo del proyecto.

También se vio como el paradigma de acoplamiento de objetos, en el que se basa el framework de comunicación construido, probó ser una herramienta efectiva de abstracción entre el desarrollo de los mecanismos de comunicación de una aplicación y el del resto de sus componentes. Esta abstracción llega al punto en que fue posible la adaptación en muy pocas líneas de código de la aplicación desde su versión mono-usuario a su versión actual multi-usuario concurrente

Finalmente se quiere mencionar que durante el desarrollo del proyecto se probaron varios escenarios para poner a prueba el acoplamiento de objetos. Si bien las pruebas solo se hicieron superficialmente y sin gran rigurosidad, la impresión obtenida fue que el acoplamiento de objetos da para ir más allá que solo la creación de aplicaciones concurrentes, y sería interesante poder evaluarlo para otro tipo de aplicaciones distribuidas ahora que se cuenta con un framework para eso.

20 Trabajos Futuros

- En la versión actual de **MOBIZ** no están incluidos todos los artefactos del estándar BPMN2.0. Un trabajo a futuro podría ser el completar la aplicación para que soporte todos los artefactos de BPMN2.0, dando pie así, a poder hacer mas y mejores pruebas de como este aporta en el levantamiento de procesos.
- Al construir un modelo BPMN este queda almacenado en forma estructurada en el servidor. Si se usa extensivamente para un mismo negocio, uno contaría con una base de datos que contiene muchísima información relevante sobre los procesos y el negocio. Por lo tanto se considera que sería una buena idea evaluar alternativas para poder administrar y sacar provecho a esta información

- Si bien en el proyecto se optó por no desarrollar aun herramientas para exportar e importar modelos desde y hacia **MOBIZ**, aun se considera que el contar con ellos puede probar ser muy útil, en especial si el resultado es compatible con algún motor de ejecución de procesos.
- En el diseño de la interfaz móvil de **MOBIZ** se consideró que debía ser posible utilizarla hasta en un *smartphone*, por lo que esta pensada para funcionar en pantallas pequeñas. Sin embargo, pese a que funciona bien con *tablets*, con pantallas más grandes, es posible que se pueda sacar más provecho de éstas si se hace una interfaz diferenciada para cada tipo de dispositivo.
- Como el análisis exhaustivo de la calidad de las interfaces desarrolladas no era parte del alcance de este trabajo, se optó por una opción más segura en base a la utilización de interfaces parecidas a otras de aplicaciones similares. Si bien se considera que el resultado obtenido es bastante bueno, se cree que se podría obtener mucha información y posibles mejoras de un análisis profundo de usabilidad en la aplicación.
- Pese a lo muy interesante que resultó ser el tema de la comunicación y la generación del framework, este no era el foco del trabajo por lo que el análisis de las funcionalidades que este debe ofrecer, no recibió todo el tiempo que merecía. Es por esto que se considera interesante la idea de hacer un estudio más acabado en esta materia.
- Si bien el framework probó ser bastante útil, se cree que lo ideal sería que, en lugar de un framework fuese un servidor completo, aunque extendible de ser necesario, y que fuese posible administrarlo solo mediante configuraciones en lugar de tener que desarrollarlo usando el framework. Por eso se cree que un siguiente paso podría ser la implementación de varios protocolos, adaptadores, puntos de entrada, etc., e incluirlos en un servidor donde se puedan escoger unos u otros mediante configuración.
- El framework da un buen punto de partida para el desarrollo de otras aplicaciones distribuidas, ya sea en HTML5 u otra tecnología. Esto puede dar pie varios proyectos de desarrollo de aplicaciones distribuidas partiendo con bastante trabajo avanzado.
- Si bien el framework **COUPLIN SERVER** y las anteriores implementaciones del paradigma de objetos acoplados, han sido utilizadas principalmente en aplicaciones con interfaces concurrentes, es posible que el paradigma sea utilizado en otros contextos. Es por eso que se considera interesante el tratar de utilizar el framework para otro tipo de aplicaciones distribuidas, usando características como el acoplamiento asimétrico, o acoplamiento sólo entre los clientes y el servidor y no entre los clientes.
- Una de las grandes sorpresas del desarrollo del proyecto fue el poco esfuerzo que requirió transformar la **MOBIZ** de su versión mono-usuario a su versión multi-usuario concurrente. Por esto se cree que esto podría extenderse a otras aplicaciones actualmente desarrolladas como aplicaciones mono-usuario.

Capítulo V

Referencias

21 Bibliografía

Business Process Model and Notation (BPMN) Version 2.0. (2011). OMG.

Baloian, N., Pino, J. A., & Jansen, M. (2007). Implementing the Coupled Objects Paradigm for Synchronizing Distributed Applications Through Firewalls.

Baloian, N., Zurita, G., Santoro, F. M., Mendes Araujo, R., Wolfgan, S., Machado, D., y otros. (2011). A Collaborative Mobile Approach for Business Process Elicitation. *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design.*

Bhaumik, S. S., & Rajagopalan, R. (2009). Elicitation techniques to overcome knowledge extraction challenges in 'as-is' process modelling: perspectives and practices. *International Journal on Process Management and Benchmarking*, 3(1).

Buelow, H., Das, M., Deb, M., Palvankar, P., & Srinivasan, M. (2010). *Getting Started with Oracle BPM Suite 11gR1.* New York: PACKT.

Gartner. (2005). *Business Process Management: Preparing for the Process-Managed Organization.*

Gonçalves, J., Santoro, F., & Baião, F. (2010). A Case Study on Designing Business Process Based on Collaborative and Mining Approaches. *Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2010).*

- Grosskopf, A., Edelman, J., & Weske, M. (Septiembre de 2009). Tangible Business Process Modeling - Methodology and Experiment Design. *Proceedings of 1st International Workshop on Empirical Research in Business Process Management (ER-BPM '09)*.
- Hoppenbrouwers, S., & Schotten, B. (2009). A Game Prototype for Basic Process Model Elicitation. *Lecture Notes in Business Information Processing*, 39(7), págs. 222-236.
- Laurentiis, G. (2011). Metodología BPM:RAD® – Rapid Analysis & Design para la modelización y diseño de procesos orientados a tecnologías BPM. En *El Libro del BPM: Tecnologías, Conceptos, Enfoques Metodológicos y Estándares*. Madrid, España: Club-BPM.
- Tewissen, F., Baloian, N., Hoppe, H., & Reimberg, E. (2000). "MatchMaker" Synchronising Objects in Replicated Software-Architectures.
- Verner, L. (Mayo de 2004). The Challenge of Process Discovery. *BPTrends*.
- Wang, H. J., Zhao, J. L., & Zhang, L. J. (2009). Policy-Driven Process Mapping (PDPM): Discovering process models from business policies. *Decision Support Systems* 48, págs. 267-281.

Capítulo VI

Apéndices

1 Anexo A. Características y Comparación entre Canvas y SVG¹⁷

1.1 Canvas

El elemento `<canvas>` es usado para dibujar imágenes en tiempo real en una página web vía JavaScript.

El elemento `<canvas>` es solo un contenedor y se para poder realmente mostrar alguna imagen se debe usar JavaScript.

En el fondo, dibujar en un canvas consiste en definir el color de cada pixel, aunque provee varios métodos que calculan los pixeles necesarios y dibuja caminos, cajas, círculos, texto o imágenes externas.

Normalmente un cambio en la imagen que muestra el canvas implica redibujar el contenido completo de este.

¹⁷Referencia:(w3schools.com)

2 Anexo B. Compatibilidad de Browser con HTML5

Tabla obtenida ejecutando el teste de <http://html5test.com/> con cada uno de los navegadores mencionados.

| | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
|-----------------------------------|-----------|-----------|----------------|----------------|-----------------------|
| Parsing rules | | | | | |
| Standards mode | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| HTML5 tokenizer | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| HTML5 tree building | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| SVG in text/html | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| MathML in text/html | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Canvas | | | | | |
| canvas element | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| 2D context | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Text | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Video | | | | | |
| video element | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Subtitle support | No ✗ | No ✗ | No ✗ | No ✗ | No ✗ |
| Poster image support | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| MPEG-4 support | No ✗ | No ✗ | No ✗ | Yes ✓ | No ✗ |
| H.264 support | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ | No ✗ |
| Ogg Theora support | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| WebM support | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| Audio | | | | | |
| audio element | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| PCM audio support | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| MP3 support | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ | No ✗ |
| AAC support | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ | No ✗ |
| Ogg Vorbis support | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| WebM support | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| Elements | | | | | |
| Embedding custom non-visible data | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Section elements | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Grouping content elements | Partial ○ | Partial ○ | Partial ○ | Partial ○ | Partial ○ |
| Text-level semantic elements | Partial ○ | Partial ○ | Partial ○ | Partial ○ | Partial ○ |
| Interactive elements | Partial ○ | Partial ○ | Partial ○ | Partial ○ | Partial ○ |
| hidden attribute | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Dynamic markup insertion | Yes ✓ | Yes ✓ | Partial ○ | Yes ✓ | Partial ○ |
| Forms | | | | | |
| input type=text | Partial ○ | Yes ✓ | Yes ✓ | Partial ○ | Yes ✓ |
| input type=search | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| input type=tel | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| input type=url | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| input type=email | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| input type=datetime | No ✗ | Partial ○ | No ✗ | Yes ✓ | No ✗ |

| | | | | | |
|-----------------------------------|------------|------------------|-----------------------|-----------------------|------------------------------|
| input type=date | No ✗ | Partial ○ | No ✗ | Yes ✓ | No ✗ |
| input type=month | No ✗ | Partial ○ | No ✗ | Yes ✓ | No ✗ |
| input type=week | No ✗ | Partial ○ | No ✗ | Partial ○ | No ✗ |
| input type=time | No ✗ | Partial ○ | No ✗ | Yes ✓ | No ✗ |
| input type=datetime-local | No ✗ | Partial ○ | No ✗ | Yes ✓ | No ✗ |
| input type=number | No ✗ | Yes ✓ | Partial ○ | Partial ○ | Partial ○ |
| input type=range | No ✗ | Yes ✓ | Partial ○ | Yes ✓ | Partial ○ |
| input type=color | No ✗ | No ✗ | No ✗ | Partial ○ | No ✗ |
| input type=checkbox | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| input type=image | Yes ✓ | Partial ○ | Partial ○ | Partial ○ | Partial ○ |
| textarea | Partial ○ | Yes ✓ | Yes ✓ | Partial ○ | Yes ✓ |
| select | Partial ○ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| fieldset | Partial ○ | Partial ○ | Yes ✓ | Partial ○ | Yes ✓ |
| datalist | No ✗ | No ✗ | Yes ✓ | No ✗ | Yes ✓ |
| keygen | No ✗ | Yes ✓ | Partial ○ | Yes ✓ | No ✗ |
| output | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| progress | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| meter | No ✗ | Yes ✓ | No ✗ | No ✗ | No ✗ |
| Field validation | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Association of controls and forms | No ✗ | Yes ✓ | Partial ○ | Yes ✓ | Partial ○ |
| Other attributes | No ✗ | Partial ○ | Partial ○ | Partial ○ | Partial ○ |
| CSS selectors | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Events | Partial ○ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Form validation | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| User interaction | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Drag and Drop Attributes | No ✗ | Partial ○ | Partial ○ | No ✗ | No ✗ |
| Drag and Drop Events | No ✗ | Yes ✓ | Yes ✓ | No ✗ | No ✗ |
| Editing elements | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Editing documents | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| APIs | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| History and navigation | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Session history | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Microdata | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Microdata | No ✗ | No ✗ | No ✗ | No ✗ | No ✗ |
| Web applications | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Application Cache | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Custom scheme handlers | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| Custom content handlers | No ✗ | No ✗ | Yes ✓ | No ✗ | Yes ✓ |
| Custom search providers | Yes ✓ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| Security | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Sandboxed iframe | No ✗ | Yes ✓ | No ✗ | Yes ✓ | Yes ✓ |
| Seamless iframe | No ✗ | No ✗ | No ✗ | No ✗ | No ✗ |
| Geolocation | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Geolocation | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| WebGL | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |

| | | | | | |
|------------------------------|------------|------------------|-----------------------|-----------------------|------------------------------|
| 3D context | No ✗ | Yes ✓ | No ✗ | No ✗ | Yes ✓ |
| Native binary data | No ✗ | Yes ✓ | Partial ○ | Partial ○ | Partial ○ |
| Communication | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Cross-document messaging | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Server-Sent Events | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| XMLHttpRequest Level 2 | No ✗ | No ✗ | Partial ○ | Partial ○ | Partial ○ |
| WebSocket | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Files | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| FileReader API | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| FileSystem API | No ✗ | Yes ✓ | No ✗ | No ✗ | No ✗ |
| Storage | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Session Storage | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Local Storage | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| IndexedDB | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| Web SQL Database | No ✗ | Yes ✓ | No ✗ | Yes ✓ | No ✗ |
| Workers | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Web Workers | No ✗ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Shared Workers | No ✗ | Yes ✓ | No ✗ | Yes ✓ | No ✗ |
| Local multimedia | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Access the webcam | No ✗ | No ✗ | No ✗ | No ✗ | No ✗ |
| Notifications | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Web Notifications | No ✗ | Yes ✓ | No ✗ | No ✗ | No ✗ |
| Other | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Page Visibility | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |
| Text selection | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Scroll into view | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ | Yes ✓ |
| Audio | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| Web Audio API | No ✗ | Yes ✓ | No ✗ | No ✗ | No ✗ |
| Audio Data API | No ✗ | No ✗ | Yes ✓ | No ✗ | Yes ✓ |
| Animation | IE9 | Chrome 17 | Mozilla 10.0.2 | Safari IOS 5.1 | Mozilla Mobile 10.0.2 |
| window.requestAnimationFrame | No ✗ | Yes ✓ | Yes ✓ | No ✗ | Yes ✓ |

3 Anexo C. Detalles de implementación CouplingServer

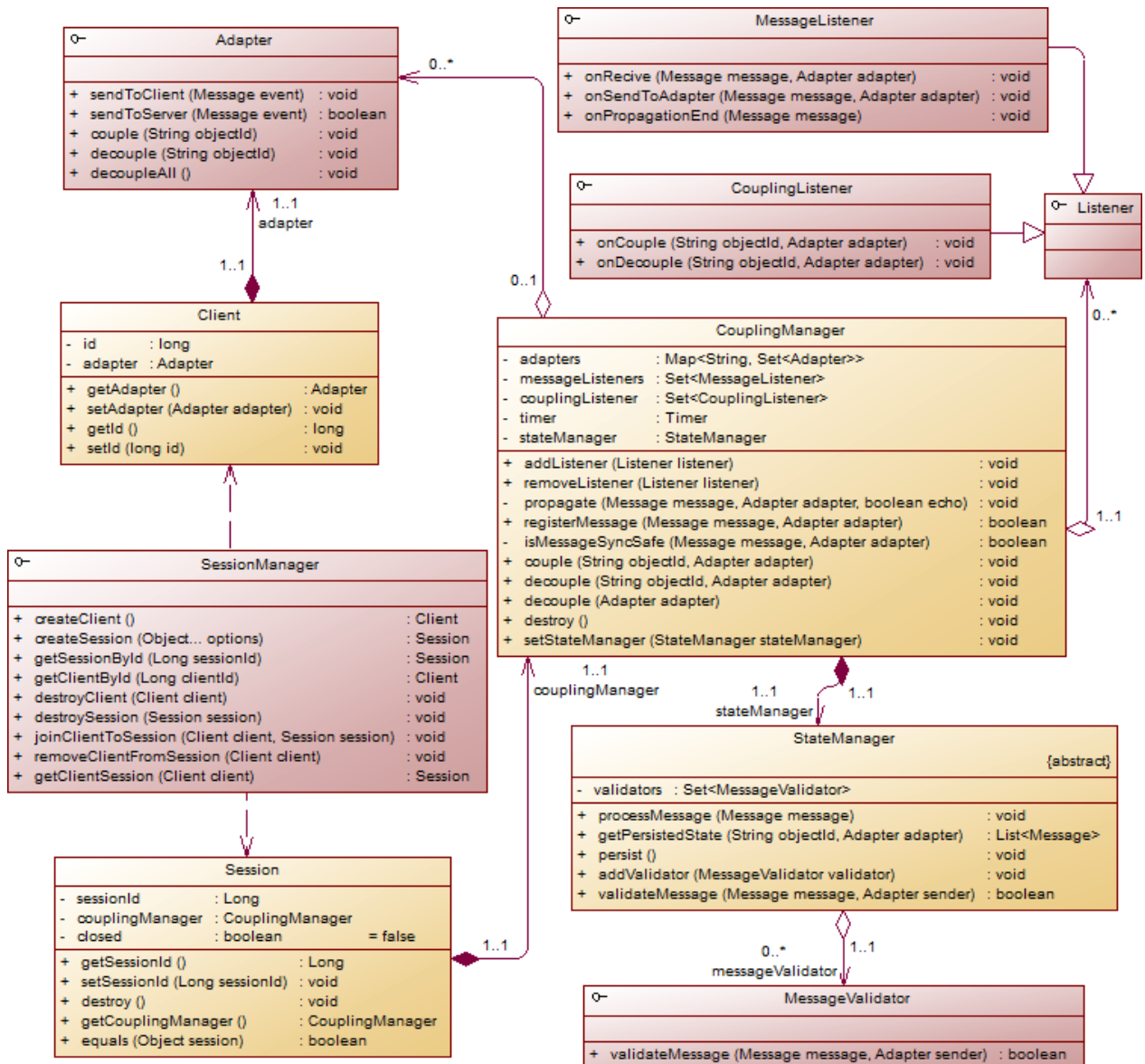


Diagrama 7: Clases CouplingServer

Como se puede ver en el Diagrama 7, **COUPLINGSERVER** esta construido alrededor de la clase *CouplingManager*, que es la clase principal del framework de comunicación y en la que ahondaremos un poco más a continuación.

CouplingManager tiene las siguientes características:

- Su tarea principal es la de recibir y procesar los mensajes, para luego reenviarlo a los clientes que correspondan, usando los adaptadores para hacerlo. Lo

interesante es que, para evitar dejar esperando a algún cliente mientras se propaga el mensaje, esto se hace asíncronamente. De esta forma, al procesarse un mensaje este queda encolado para ser propagado y persistido mas adelante.

- Tiene dos métodos marcados como *synchronized* para el control de la concurrencia, que son el que se encarga de recibir las llamadas y el que se encarga de propagarlas. De esta forma se transforma en el semaforo ideal por lo que es utilizado también por su *StateManager* para manejar su concurrencia mediante el bloques del tipo *synchronized(object){}*.
- Cuando llega un mensaje, antes de encolarlo, puede validarlo usando su *StateManager*, quien a su vez puede tener una serie de validadores que serán finalmente los encargados responder si el mensaje es o no valido. De no ser valido el mensaje se descarta y el método responde *false*. Esta respuesta debe ser propagada hasta el cliente que envía el mensaje quien debe honrarla y descartarlo. Por esto solo se pueden validar mensaje del tipo evento, pues para generar el estado se debe realizar el cambio antes de enviar el mensaje, lo que evita que se pueda descartar ante una respuesta negativa del servidor.
- Si dos mensajes enviados casi simultáneamente al servidor por dos clientes distintos modifican el mismo elemento, esto podría causar que, dado el orden de ejecución, lo que un cliente vea sea distinto del otro. Para manejar este caso se procede de la siguiente manera:
 - Cuando llega un mensaje se evalúa si tiene o no riesgo de que cause conflicto con otro mensaje que aun no ha sido enviado. Esta evaluación puede ser mas estricta, que asume que cualquier par de mensajes encolados podrían causar conflicto, o puede ser menos estricta, donde solo mensajes para el mismo objeto son conflictivos.
 - Si hay conflicto se procede normalmente, de haber conflicto el método responde *false*, pero el mensaje no se descarta, sino que se encola pero se marca como mensaje con eco. Un mensaje con eco, a diferencia de es enviado también al cliente que lo generó. De esta forma, de ser un evento, este no se ejecutara al ser generado sino al ser recibido como eco, asegurando así el orden de ejecución. De ser un cambio de esta no se puede descartar, pero eso no es problema pues un cambio de estado adicional antes, que es el original, no genera problemas.